IPP-Report

Max-Planck-Institut für Plasmaphysik

Alexander van Roessel

Convolutional Neural Networks for the In-Situ Investigation of Blistering on Plasma-Exposed Metal Surfaces

IPP 2021-05
Februar 2021

Convolutional Neural Networks for the In-Situ Investigation of Blistering on Plasma-Exposed Metal Surfaces

A Master Thesis in Physics by

Alexander van Roessel

Technical University of Munich
Department of Physics
01.12.2020

Primary Reviewer:	Prof. Dr. Ulrich Stroth
Secondary Reviewer:	Prof. Dr. Rudolf Neu
Primary Supervisor:	Dr. Armin Manhard
Secondary Supervisor:	Dr. Udo von Toussaint

Abstract

For the inner wall material of a future fusion reactor, typically low-sputtering metals such as tungsten are considered. However, due to the interaction with energetic hydrogen particles from the plasma, the surface may be altered. In some cases, especially for metals with an endothermic heat of solution, hydrogen-filled bubbles can emerge on the surface of the metal. These features are called blisters and can influence the retention of hydrogen and its diffusion into the material, which raises concerns especially with regards to the inventory of radioactive tritium in the wall. So far, the investigation of blistering phenomena was limited to before-after comparisons and required the manual identification and characterization of blister features. This allowed only qualitative analysis of a small number of blisters. These limitations prevented a thorough investigation and deferred the understanding of the underlying processes of blister formation. For the first time, a large-scale and time-resolved investigation of blisters is conducted in the present thesis. This involves the in-situ observation of a plasma-exposed molybdenum sample with a high-resolution camera and the automated evaluation of the acquired images with appropriate techniques. For the observation of the sample, a pre-existing experimental setup (called PlaQ) is used, while for the automated evaluation of the image data, a sophisticated procedure is developed that comprises, among other things, the application of three separate convolutional neural networks. These networks perform the main task of localizing and characterizing individual blisters and were trained with artificially generated images. The developed framework performs well especially for low blister densities and makes the blister identification process transparent and reproducible for the first time. In the course of the present thesis, several investigations on a molybdenum sample were conducted. A study of the abundance and surface coverage of blisters reveals a distinct saturation effect. The analysis of the growth dynamics of blisters indicates that blisters exhibit larger growth rates when they emerge at higher fluences. In addition, the outcome of a nearest-neighbor analysis of the spatial distribution of blisters on the surface suggests that blisters exhibit no discernible medium- to long-range impact on their surroundings. The results obtained by these investigations indicate that the in-situ and large-scale investigation of blistering is feasible and can contribute to a better understanding of blistering phenomena and the underlying physics. In addition, the techniques developed in the present thesis can be utilized for the consistent analysis of other sample materials or applied to different experimental scenarios.

Contents

Chapter 1

Introduction

Climate change constitutes one of the biggest challenges of our time. Its main cause is the carbon dioxide (CO_2) that is emitted in the context of modern human civilization [1]. The majority of 70% of these CO_2 emissions originate from the energy sector [2]. The global energy demand is expected to grow by up to 60% in the 2010-2050 period [3] due to a rising population and an increasing living standard. Hence, there is a desperate need for sustainable energy sources which also need to be clean, secure, and reliable. A promising technology that can satisfy these demands is nuclear fusion. It strives to harness the energy that is released by the fusion of the atomic nuclei of deuterium (D) and tritium (T). For this reaction to happen efficiently, extreme conditions are required that can for example be achieved in a hot plasma.

One of the two main designs for a potential fusion reactor is the so-called Tokamak, which has a toroidal shape and is operated in pulses. The ionized particles in the fusion plasma are confined within a vacuum vessel by strong magnetic fields. The components inside the vessel and in particular the material of the inner vessel wall are exposed to high fluxes of energetic particles and have to withstand extreme conditions. Therefore, metals with a low sputtering yield, a high melting point, and good thermal conductivity, such as tungsten, are currently considered as material for in-vessel components and the plasma-facing wall. Since the fusion plasma will mainly consist of hydrogen isotopes, the material will be exposed to large fluences of hydrogen atoms and ions with various impact energies. Thus, the interaction of these energetic particles with the material and in particular the transport and retention of hydrogen isotopes therein are of great relevance. One of the major issues is the retention of radioactive tritium in the wall since it would pose a serious safety concern if large amounts of tritium were retained in the vessel wall [4].

During the interaction of hydrogen isotopes with metals under conditions as they are expected to occur in a fusion chamber, gas-filled cavities, which are also called blisters, can arise near the surface [5]. These features influence the diffusion and retention of hydrogen isotopes in the material, including tritium, and thus it is very important to investigate the emergence of blisters as well as the underlying processes. So far, no comprehensive theory regarding the formation and evolution of blisters and their influence on the diffusion and retention of hydrogen has been developed. In particular, researchers have not yet been able to study the dynamics of blister evolution in detail, but only considered isolated and static observations of blisters before and after plasma exposure. However, the time-resolved and in-situ investigation of a large number of blisters yields additional information about the development of blisters and may thereby allow to derive conclusions about the underlying processes. Such investigations are advantageously realized by means of an automated analysis procedure that provides stable identification, tracking, and characterization of blisters. A promising approach exploits the fact that the surface morphology formed by blisters can be observed with an optical system that provides sufficient resolution. This would involve the observation of a sample with a high-resolution camera while it is exposed to a plasma and the evaluation of the resulting images with appropriate techniques. An appropriate camera system was recently added to the experimental setup [6] described in Section 4.1, which allows to perform in-situ video analysis of a specimen during plasma exposure. However, no appropriate methodology for the automated evaluation of the acquired images has been developed so far. For the solution of such generic image recognition tasks, convolutional neural networks (CNNs) have proven to be a suitable and efficient technique.

The present work will employ an ensemble of different CNNs to address the above stated challenges in the study of blistering phenomena.[1] The specific goal of the present work is to develop automated routines that can identify and localize blisters, estimate important blister parameters such as their size, and track individual blisters over time. Thereby, it aims at providing a comprehensive framework for the automated evaluation of image data to enable large-scale dynamic investigations of blistering processes. This framework will allow researchers to gain information on the behaviour of individual blisters as well as acquire statistics on aggregated blistering parameters. It will provide the means by which other researchers can examine additional samples consistently while making the identification procedure of blisters transparent and reproducible. This will contribute to the understanding of blister formation and the underlying processes.

The following chapters will give a detailed description of the data acquisition, the preprocessing, and the implementation of the automated detection with CNNs. Also, an evaluation of the detection quality and results on image data from a molybdenum (Mo) sample will be presented. The results of these analyses indicate a high reliability of the detection for those blister densities it was adjusted to and yield first insights into the dynamics of blister formation and the potential interaction of blisters among each other.

[1]All algorithmic routines in the present thesis were implemented in the Python programming language and will be provided in a GitLab repository [7] along with an appropriate virtual environment. Additionally, the source codes of the most important components are given in the appendix.

Chapter 2

Blistering on Plasma-Exposed Metal Surfaces

The purpose of this chapter is twofold. On the one hand, a general introduction to the interaction of hydrogen with metals and the state of research concerning blistering on plasma-exposed metal surfaces will be given. This introduction will also include the identification of open research questions and the outline of methodological requirements for their investigation. On the other hand, this chapter will provide an overview of the approach that is used to fill some of the gaps in the current state of research and defines the goals this thesis strives to achieve. To accomplish both these purposes, the following sections will first give a brief outline of the absorption and diffusion processes of hydrogen in metals. Afterwards, the interaction of hydrogen with lattice defects will be discussed. Subsequently, the hydrogen-induced blistering of metal surfaces and its influence on the retention and diffusion of hydrogen in the material will be treated. The last section will outline open research questions and how they will be addressed in this thesis.

2.1 Hydrogen Absorption and Diffusion in Metals

One of the first researchers to study the energetic properties of hydrogen molecules and atoms near metal surfaces was Lennard-Jones. As a result of his studies, he proposed a schematic description of the potential energy relations in the vacuum, near the surface, and in the bulk of metals [8]. An illustration of these relations is given in Fig. 2.1. In this model, the basic state of two hydrogen atoms is assumed to be an isolated H_2 molecule in vacuum and hence the energy for this state is defined as zero. This means that an isolated hydrogen atom has a potential energy of 2.25 eV, which is equivalent to half the dissociation energy of the molecule. The energy curves of both the hydrogen molecule and the hydrogen atom have minima before the surface, the depth of which are labeled with Q_P and Q_C. These are the heat of physisorption in the case of the molecule and the heat of chemisorption in the case of the atom, respectively. Close to the surface, the energy of the isolated hydrogen atom is much lower than for the molecule and the intersection of the two energy curves indicates the distance at which the molecule will dissociate. The activation energy that is necessary to achieve dissociation of the molecule and allow the two dissociated atoms to chemisorb individually is denoted by E_C. It is much lower than the dissociation energy of the molecule in vacuum. In order to be absorbed by the metal, the hydrogen atoms must be, aside from Q_C, provided with an additional energy E_{sb}. This energy, which typically is a few eV, allows the atom to penetrate the surface and diffuse into the bulk. Kinetic ions, for example originating from a plasma source, can surpass this energy barrier and directly penetrate the surface and diffuse into the bulk. They lose their energy by elastic collisions with the lattice atoms as well as friction with electrons until their energy is insufficient for further propagation. The ions or atoms finally come to a halt at an interstitial site or get trapped in a defect (cf. Fig. 2.1). Ebisuzaki and O'Keefe [9] proposed that the dissolved hydrogen contributes its electron to the conduction band of the metal, which has for example been verified by Zamir [10], and thereafter exists as proton. The authors further argue that the positive potential of the proton is screened by an increased local electron density.

Thermodynamically, the absorption and desorption of hydrogen in the metal represents a two-phase system. On the one hand, there is hydrogen in the gas phase and on the other hand there is hydrogen that is absorbed into the metal. In equilibrium, the processes of dissolution in and desorption from the metal balance each other. The equilibrium hydrogen concentration C in the bulk is described by Sieverts' law [12]:

$$C = S\sqrt{p} \,, \tag{2.1}$$

where p is the pressure of the gas phase and S is the solubility of hydrogen in the metal for a given temperature T. It is applicable only for conditions under which the hydrogen gas phase can be treated as an ideal gas and the hydrogen concentration is small enough so that hydrogen-hydrogen interactions or hydride formation can be neglected [13]. The solubility is given by

$$S = S_0 \exp\left(-\frac{E_S}{k_B T}\right) \,, \tag{2.2}$$

where E_S is the heat of solution of hydrogen in the metal, which is illustrated in Fig. 2.1, and S_0 is just a constant. For metals with large heat of solutions, such as tungsten (1.1 eV) or molybdenum (0.54 eV) [13], the uptake of hydrogen from the molecular state is endothermic and these metals will exhibit smaller hydrogen solubilities and concentrations than those with a lower heat of solution. However, when regarding the atomic state, the uptake of hydrogen is exothermic for metals with larger heat of solutions as well.

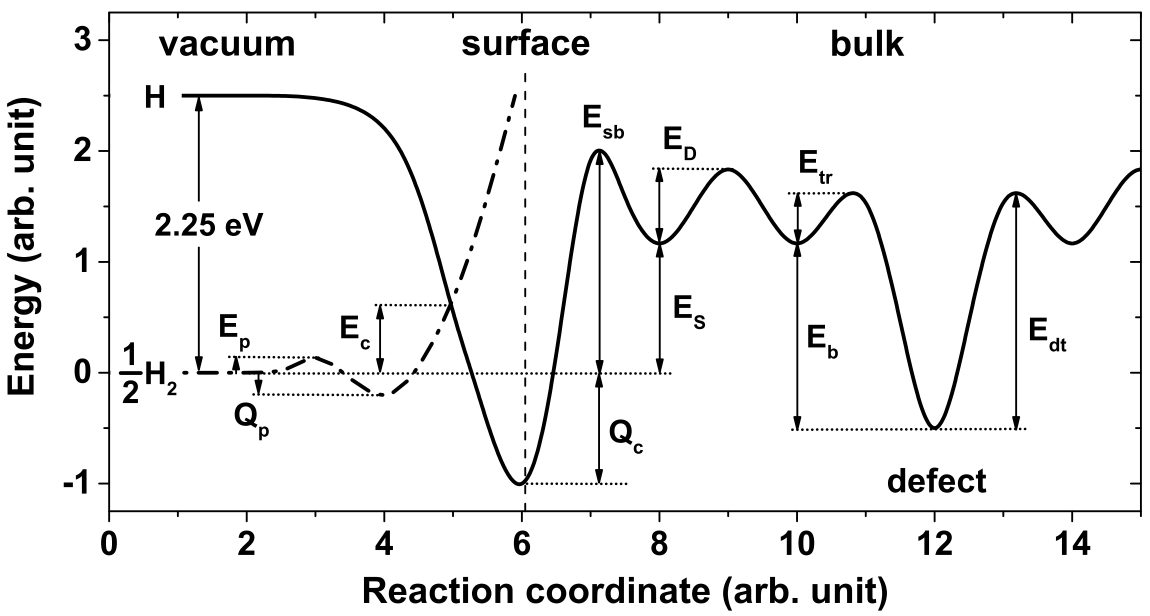


Figure 2.1: *Illustration of the potential energy of atomic (full line) and molecular (dashed line) hydrogen in the vacuum, at the surface of a metal, and in its bulk. The heats of physisorption and chemisorption are denoted by Q_p and Q_c and the respective activation energies by E_p and E_c. E_s is the heat of solution, while E_{sb}, E_D, E_{tr}, and E_{dt} are the activation energies for absorption, diffusion, trapping, and detrapping respectively. E_b is the binding energy of a hydrogen atom in a defect. The figure was kindly provided by Mikhail Zibrov [11].*

2.2 Hydrogen Interaction with Defects

The discussion above regards the case of a single-crystalline, defect-free metal lattice. However, any real sample has some defects in its structure due to finite temperatures. In addition, some imperfections are caused by the mechanical manipulation during the manufacturing process. Among these defects are geometrical aberrations such as edge dislocations and screw dislocations, impurities consisting of atoms of another element, and grain boundaries. Some of these defects are illustrated in Fig. 2.2. Apart from these intrinsic alterations of the lattice structure, defects can also be introduced for example by the irradiation of the metal with ions of sufficiently high energy. These in particular include vacancies, vacancy clusters, and voids. These defects represent energetically favorable positions for the hydrogen which will consequently start to accumulate at these locations [14, 15].

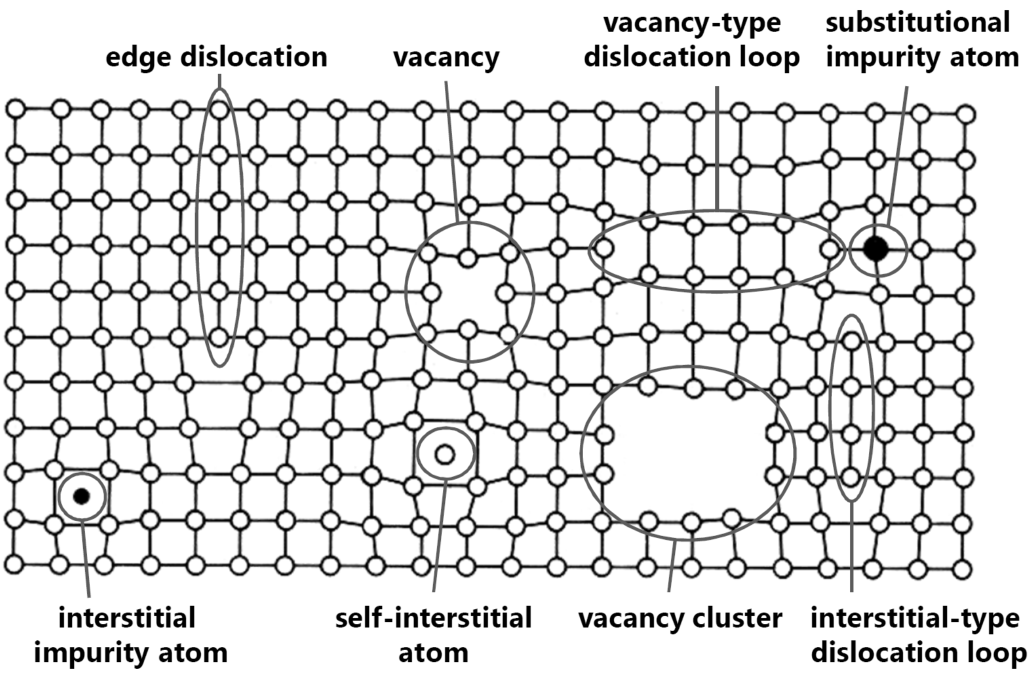

edge dislocation
vacancy
vacancy-type dislocation loop
substitutional impurity atom
interstitial impurity atom
self-interstitial atom
vacancy cluster
interstitial-type dislocation loop

Figure 2.2: *Various types of defects in metal lattices. Especially in voids caused by vacancies or vacancy clusters, hydrogen can accumulate and leads to blister formation. Figure adapted from [11].*

2.3 Hydrogen-Induced Blistering

Metals with an endothermic heat of solution for hydrogen like molybdenum and tungsten can be easily oversaturated, for example by exposing them to a deuterium plasma. This leads to phenomena that are not observed in experiments with gas loading. As described before, the energetic ions can directly penetrate the surface and start diffusing into the bulk. The concentration of hydrogen at a certain depth is then governed by the incoming flux and the diffusivity. For ion energies of just a few eV, the corresponding chemical potential suffices to allow for very high concentrations which can cause extraordinarily high equilibrium pressures as estimated by Eq. (2.1). The relation does not hold exactly since the gas cannot be assumed to behave as an ideal gas under these conditions, however thorough calculations of this have been conducted by some researchers [16]. At some defect locations, the hydrogen starts to form molecules [16] and the pressure of the hydrogen gas may become high enough for the stress at the cavity boundaries to rise far beyond the yield strength of the metal [17]. This can cause an active displacement of lattice atoms and thereby increase the volume of the cavity [18, 19]. These defects therefore cannot be saturated and will continue to grow, which leads to the formation of hydrogen gas bubbles in the material [5]. When the cavity lies close to the plasma-exposed surface, the displacement may

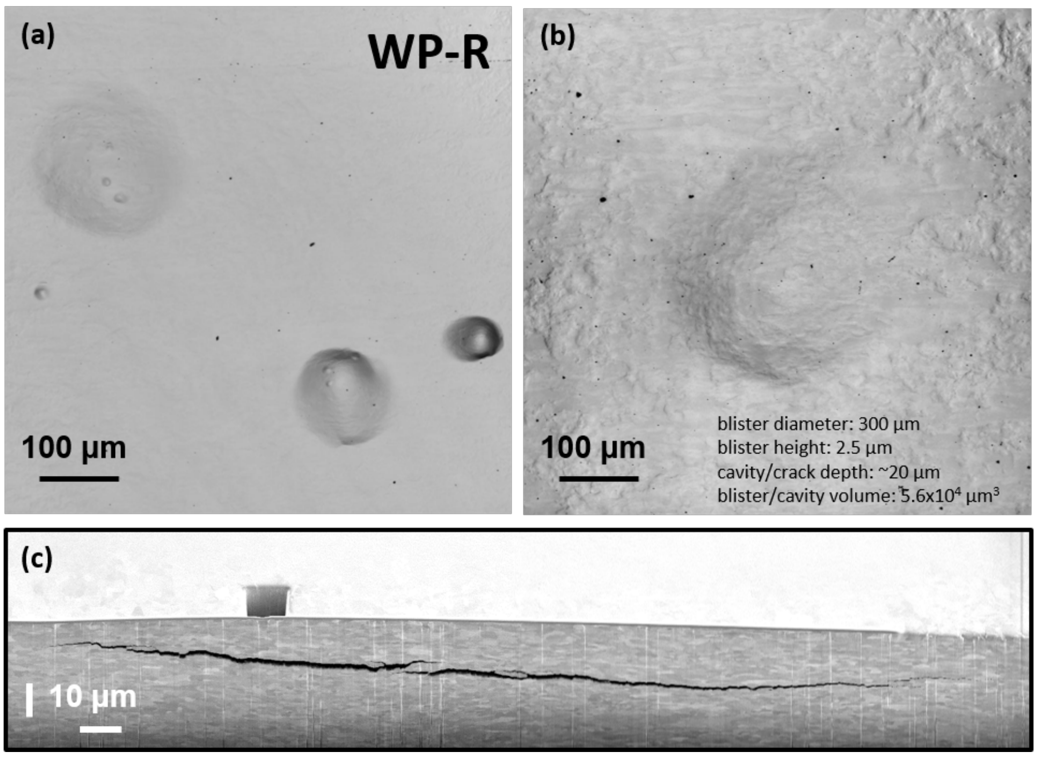


Figure 2.3: *Sample images of some blisters on rolled tungsten. Panels a) and b) were obtained by confocal laser scanning microscopy. In panel b), detailed information on the parameters of the blister that is visible is given. Panel c) is a scanning electron microscope cross-section view of a blister that was degassed before cross-sectioning by puncturing it with a focused ion beam. The cross-section is horizontally tilted by 38° with respect to the viewing plane. Images kindly provided by Martin Balden.*

become visible on the surface. These features take the form of blister-like structures [20, 21] and are therefore called blisters. Some sample images of blisters, including cross-sectional views, are given in Fig. 2.3.

Important blister parameters such as their size and abundance can be very different depending on the experimental setting. These parameters in particular depend on the material, the implantation temperature, the fluence[1], and the microstructure of the specimen [22]. For example, it has been shown that low fluxes with much longer exposure times correlate with fewer but larger blisters. As explanation it is put forward that hydrogen can reach larger depths and accumulate at weak grain boundaries [23]. At the same time, the size and density of blisters strongly increases

[1]The fluence is the flux density integrated over time and typically measured in incoming deuterons per square meter, i.e. Dm^{-2}.

with fluence [23]. The formation of blisters can be reduced or suppressed entirely at high temperatures [18] and on rough surfaces [24]. Blisters can also vanish due to degassing, for example during thermal desorption spectroscopy or when their cap is penetrated [17, 25]. However, after degassing, the deformation around the grain boundary that is caused by the inflation of the blister cannot be recovered elastically.

In the discussion so far, it has not been distinguished between the different hydrogen isotopes, i.e. protium, deuterium, and tritium. In fact, their interaction with metals is likely to be very similar in most regards. However, there is an important difference with regards to the diffusivity. Deuterium and tritium usually have a lower diffusivity than protium due to their larger mass. This general mass dependence of diffusion velocity is in fact used for isotope fractionation [26]. From this effect it follows that the hydrogen concentrations below the surface, in particular in the region where blistering occurs, will be higher for the heavier isotopes given a certain flux. Therefore, it is reasonable to assume that blistering features occur at lower fluences for deuterium and tritium.

2.4 Influence of Blisters on Hydrogen Retention

It has been shown by A. Manhard that blisters significantly increase the retention of deuterium [22], which takes place in or close to the cavities [23]. The spatial distribution of the deuterium inventory depends on the size and shape of the blister and hence on several experimental settings. On the one hand, in large and round blisters, up to half of the deuterium inventory can be stored as D_2 gas inside the main cavity of the blister. On the other hand, in smaller and more flat blisters, a substantial fraction of the deuterium can apparently be retained in lattice defects localized around the blister [27], which stands in contrast to what was suspected for example by Causey *et al.* [28]. These lattice defects are induced by the expansion of the cavity [29, 30], which is caused by the high pressure of the D_2 gas. Manhard supposes that dislocations emitted in the stress field of the crack-tip [31] may be the main reason for the increased deuterium retention. Overall, blisters and blister-like structures can amplify and even dominate other factors affecting the hydrogen retention such as the microstructure [22]. At the same time, Bauer *et al.* have shown that near-surface blisters with open caps increase the hydrogen reemission rate and thereby reduce the hydrogen transport into the bulk [32]. In their investigation, the deuterium flux into the bulk was reduced by 60% compared to the unblistered case. In summary, the occurrence of blisters has an ambiguous effect on the retention of hydrogen. More research is required to clarify the role of blisters in the build up of hydrogen inventories and in the reduction of the permeation flux.

2.5 Open Research Questions

So far, no comprehensive theory regarding the formation and evolution of blisters, the production of additional defects, and their role in the retention and diffusion of hydrogen has been developed. For example, it has not yet been confirmed whether blisters increase the probability for the occurrence of further blisters nearby, i.e. clustering. On the other hand, on vanished blisters less new blisters may occur because the cap may act as a degassing channel causing the deuterium to escape and preventing the build up of high pressures. In addition, the dynamics of their evolution have not been studied in detail so far.

To thoroughly investigate the occurrence and behavior of blisters, an extensive parameter study needs to be conducted. It should comprise variation on the experimental parameters such as the type of hydrogen isotopes in the plasma, the temperature, and the ion energy. In addition, sample parameters like the type of preparation and the material should be varied independently. The analysis of the sample should ideally provide time-resolved information on blister abundance, locations, and sizes. So far, only selective and isolated observations of blisters were made where the specimen was analyzed statically before and after plasma exposure. This limitation can be resolved with the experimental setup described in Section 4.1, which allows to perform in-situ video analysis of the specimen during plasma exposure. This enables a detailed and time-resolved study of individual blisters. However, to enable meaningful statistical statements about blister parameters, the identification, tracking, and analysis of a large amount of blisters over a time-series of images is necessary. Additionally, this needs to be done for a variety of samples subjected to different experimental conditions and the analysis process should ideally be stable, transparent, and reproducible. This can only be accomplished by means of a blister analysis framework that is automated to a great degree. This thesis strives to achieve this by exploiting the fact that the surface morphologies formed by blisters can in many cases be observed with an appropriate optical system (cf. Section 4.1). The evaluation of the acquired images represents a classic image recognition task. This problem will be addressed with convolutional neural networks, which will be introduced in the next chapter.

Chapter 3

Computer Vision with Convolutional Neural Networks

The purpose of this chapter is to give an introduction to the methodology used in the present thesis and to explain why its application is appropriate. The first two introductory sections will give an overview of the topic of artificial neural networks in general and of its history. Subsequently, the setup and functional components of such networks will be discussed. Section 3.7 will give a detailed outline of how neural networks are trained while Section 3.8 presents regularization techniques that ensure the proper functioning of the networks. The specific method of convolutional neural networks (CNNs) along with it's major features will be treated in depth in Section 3.9. Finally, Section 3.10 will outline why this method is in particular suitable for the analysis in the present thesis.

3.1 Subsumption Under the Machine Learning Landscape

Traditionally, the automated analysis of data and the derivation of information for the support of decision-making processes involved the application of expert-designed rule systems. This approach is suitable for some applications, in particular those where humans have a good understanding of the task or the complexity and required flexibility is low. However, many relevant problems in automated data analysis do not fall into this category. For example in the case of image recognition, the computer may perceive images and objects differently than humans. It remains contentious to what degree the perceptive processes of state-of-the-art image recognition tools like CNNs match the process in the visual cortex of humans [33–35]. Hence, it is difficult for a human to come up with a good set of rules that describes the object or the process of object recognition well. For this kind of problems, computers must be able to learn from data without relying on a human to program every possible case more or less explicitly. This is the realm of machine learning, which can be defined as "[. . .]

algorithms that can learn from data without relying on rules-based programming." [36]. It is used to address tasks as diverse as regression, forecasting, classification, clustering, image recognition, and artificial intelligence and employs a large variety of techniques ranging from simple linear regression to artificial neural networks. For each technique, there can be a range of different algorithmic implementations. The multitude of techniques and algorithms that are subsumed under machine learning can be categorized along the following criteria: whether or not the system is trained with human supervision and known outcomes (supervised vs. unsupervised learning); whether or not they can learn incrementally on the fly (online vs. batch learning) and whether they just compare new data to known data points (instance-based learning) or instead detect patterns in the training data and infer a predictive model (model-based learning). For more details on this categorization, see the book on machine learning by Géron [37].

This thesis deals with the automatic detection of blisters in images and therefore falls into the realm of computer vision, which can be defined in the following way: "Computer vision is the construction of explicit, meaningful descriptions of physical objects from images." [38]. This general definition can be divided into several specific tasks, as was done for example by Voulodimos *et al.* [39]. A good introduction to some major tasks is also given in a course by the Stanford University School of Engineering [40]. One important class of problems involves the identification of several individual object instances in the same image. This kind of problem is called object detection [41] and plays an important role in applications like autonomous driving. In this case, several parameters are of interest: the total number of object instances, the classes these instances belong to, their location in the image, and additional parameters like their size. Since a major aim of this thesis is to develop a tool for the automated detection of blisters in images, it represents a classic object detection task. A state-of-the-art method for addressing this kind of problem are CNNs [39, 42]. This versatile technique will be applied in a supervised setting for the analysis in the present thesis (cf. Section 3.9). Model-based methods and a combination of online as well as batch learning techniques will be applied. The details of this approach will be discussed in the following sections and Chapter 6.

3.2 History of Artificial Neural Networks

In 2006, Hinton *et al.* published their paper on the recognition of handwritten digits via a deep neural network [43]. In the aftermath of this breakthrough paper, deep learning as well as machine learning in general have drawn a lot of attention and have since developed rapidly. They are now powering many techniques and features in the processing of information in the digital world. Deep learning has become a synonym for machine learning with artificial neural networks in general but originally referred to networks with a specific setup only. This setup would comprise a layered structure with at least two so called hidden layers, which will be explained further below. However, this structure has become prevalent in most of today's networks.

In the realm of artificial intelligence, intelligence is understood to be a rather broad concept and not limited to specific human capabilities. Its main components encompass learning, reasoning, problem solving, perception, and the use of language [44]. When scientists first began to try to build intelligent machines, it was natural to look for inspiration in nature, since this has often initiated the development of innovations. The brains of humans and animals perform incredibly difficult and complex tasks and can adapt to a wide array of problems. Hence, it was logical to let natural neural networks that make up these brains serve as a blueprint for the design of artificial neural networks (ANNs) in order to create a general problem-solving machine. The supplement "artificial" will be dropped in in the following to improve readability. Generally speaking, ANNs are meant to provide an approximation of information processing in biological systems. In natural brains, the neurons are connected to each other via synapses and can be either inactive or activated. In computer science, the neurons are represented by computational units and the synapses are equivalent to the mathematical input and output relations between units.

The theoretical foundation for ANNs was laid by various authors [45–47]. First attempts to realize ANNs were made as early as 1960 [48]. Since then, the attention given and resources allocated to this field of research have experienced several ups and downs, some of them being nicely described in chapter 10 in Ref. [37]. In recent years, interest and research activity have grown exponentially and are very likely to continue doing so in the future. There are two major trends driving this development. On the one hand, the availability of both training data and computational resources has increased rapidly. The former allows to train ever more complex networks and the latter enables to conduct their training in a reasonable amount of time. On the other hand, the development of open source libraries such as Tensorflow and PyTorch as well as user-friendly high level APIs such as Keras have increased the deep learning community and boosted the development of new networks and their application in industry.

3.3 Artificial Neurons: The Basic Building Blocks

What gives neural networks their name are their fundamental computational units, the so called neurons. Each neuron has the same structural components, which comprise the weighting and summation of the inputs, the application of a certain activation function, and the transfer of the output to the subsequent layer. Different network architectures only differ in the arrangement of these units. Fig. 3.1 shows

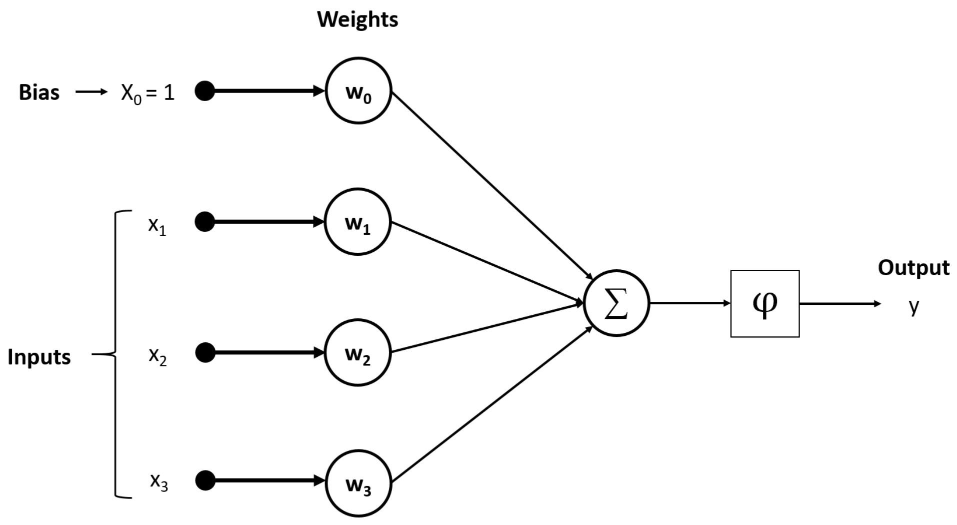


Figure 3.1: *Illustration of the working principle of a single artificial neuron. The inputs are multiplied by individual weights and summed up. The activation level of the node that represents the bias has a fixed value of one. The result is subjected to the activation function which then determines the final output activation level.*

the basic components of a single neuron. It can be mathematically described by the following equation:

$$y = \varphi\left(\sum_{j=0}^{n} w_j x_j\right). \tag{3.1}$$

The inputs x_j can represent any information as long as it can be represented in numbers. Those inputs with j = 1, ..., n are multiplied by individual weights w_j. An additional input node is used to function as an anchor for the bias, which is represented by an additional weight w_0. The activation value of this anchor node is fixed at 1. The sum of the weighed inputs including the bias term is the overall input to the neuron. Given this input, the activation function φ determines the response y of the neuron. This response is called the activation level of the neuron.

3.4 Multi-Layer Neural Networks

In order to solve complex computational problems, the combination of a large number of neural units is necessary. Over time, this lead to the development of neural networks. A standard reference that covers all basic aspects of neural networks with great diligence is the book "Neural Networks for Pattern Recognition" by Christopher M. Bishop [49]. A common way of arranging the large number of units in ANNs is in successive layers. This layered structure has become prevalent in modern neural

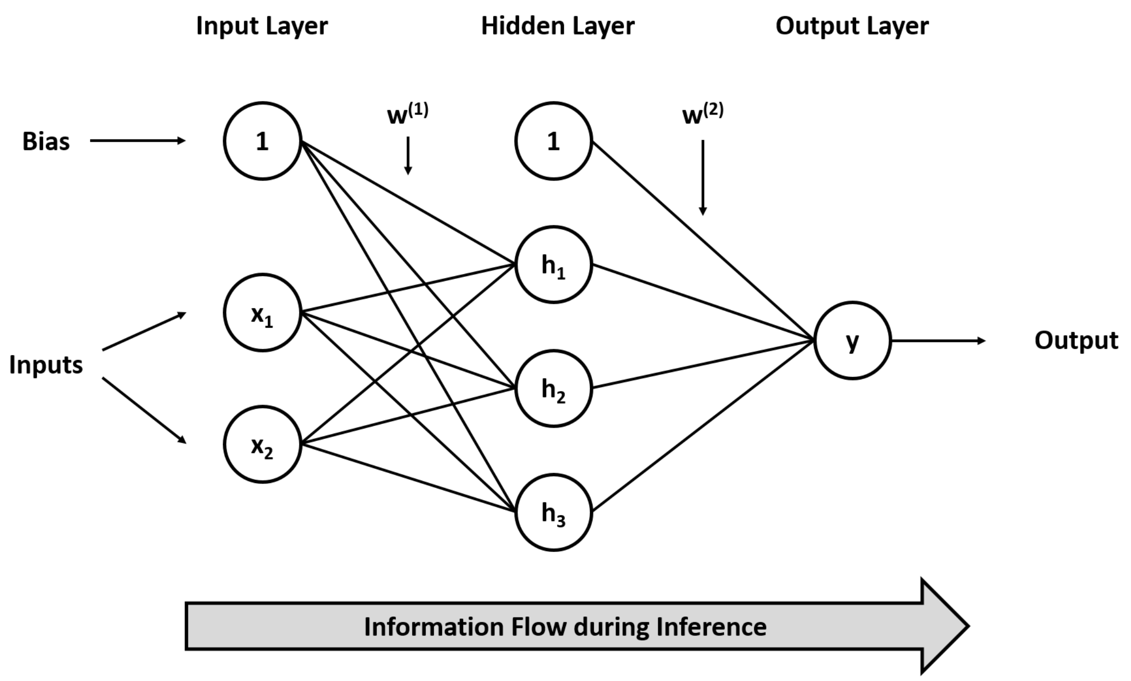


Figure 3.2: *Network diagram for a feedforward multi-layer ANN with one hidden layer. The neural units of the input, hidden, and output layers are represented by nodes. The links between the nodes represent the weight parameters and the information flows from left to right during inference. The biases are included in the network diagram by additional nodes with a fixed output value of 1. The individual bias parameters are represented by links coming from these anchors and are denoted by $w_{i,0}$. The $w^{(n)}$ denote all the weights between the layer n and the successive layer. The overall network function that corresponds to this network is given in Eq. (3.2).*

networks. They were first investigated under the term multi-layer perceptron [50] by Frank Rosenblatt [46]. However, modern neural networks usually make use of continuous nonlinear activation functions instead of discontinuous activation functions as in these first setups. This difference is of major importance because it means that the overall network function is differentiable with respect to the network parameters. This feature will be vital for the application of training algorithms, which are discussed in Sections 3.7.3 and 3.7.4.

There are different ways of defining the number of layers in a neural network. In this thesis, the convention of counting the number of computational stages that are applied in the network will be used. An example of a simple two-layered network is depicted in Fig. 3.2. In general, ANNs such as this one consist of an input layer, an output layer, and an arbitrary number of hidden layers in between. The neurons are represented by nodes and the input and output relations are represented by links. In feedforward networks as employed in this thesis, the information flows from the input to the output layer. Nodes in one layer combine and alter the information contained in the outputs of the previous layer and pass it on to the next layer. There are also recursive NNs in which the information can also be passed to preceding layers. If all neurons in a layer are connected to all neurons in the previous layer it is called a dense or fully connected layer. The weights, biases, and activation functions can be chosen individually for every neuron or neural connection respectively. Hence, the number of free parameters in the network can become very large which allows for a high complexity of the model that the network represents. The terms network and model are often used synonymously and will so in the following. The weights and biases are optimized during training of the network (cf. Section 3.7). The activation function is typically the same for all the hidden units in the network. Its type is a hyperparameter that has to be set prior to training. Hyperparameters and their tuning are discussed in detail in Chapter 6.

Every neural network can mathematically be described by a nonlinear function from a set of input variables x_i to a set of output variables y_k. They can be viewed as generalizations of linear models that use multiple stages of processing to compute the outputs. In order to get the overall network function, one has to apply all stages of computation to the inputs successively. For networks with more than one hidden layer, the overall network function will have a nested structure which is reminiscent of the layered structure of the network. For the case of the simple two-layer network from Fig. 3.2, the overall network function for a single output neuron can be written as

$$y_k(\mathbf{x}, \mathbf{w}) = \varphi^{(2)}\left(\sum_{j=0}^{M} w_{kj}^{(2)} \varphi^{(1)}\left(\sum_{i=0}^{N} w_{ji}^{(1)} x_i\right)\right) \quad . \tag{3.2}$$

Here, $\varphi^{(n)}$ is the activation function at layer n of the network, N and M are the number of nodes in layer one and two and $w_{\alpha\beta}^{(n)}$ are the respective weights between nodes in layer n and the following layer. In the general case, the overall network function would imply the successive application of all the computational transformations of all layers. Its evaluation can be viewed as a forward feeding and stage-wise modification of information through successive layers. This is why ANNs are also called feed-forward neural networks. The result that is obtained when evaluating the overall network function represents the prediction of the neural network under the current weights. The training process of a neural network involves many evaluations of the overall network function and appropriate incremental improvements to the network parameters (cf. Section 3.7.4).

3.5 Activation Functions: Choosing Neural Response

The layered structure of neural networks results in multiple successive applications of non-linear activation functions. Thereby, the overall network function becomes nonlinear. This is important because otherwise the network function could be reduced to a single linear transformation and the network would hence loose its computational power. Apart from their nonlinearity, activation functions can have other important features which heavily influence the behaviour of the network. Therefore, the type of activation function is an important choice to make when setting up a neural network. There is a small number of activation functions which are employed in the majority of use cases. These include the linear activation function, the logistic sigmoid, and the hyperbolic tangent functions as well as the so called rectified linear unit (ReLU) function. Linear activation functions are employed in the output layer for some regression tasks. The definition of the ReLU function is given in Eq. (3.3) and all of the functions are illustrated in Fig. 3.3. Out of the four depicted functions, the sigmoid and the ReLU function will be employed in the course of this investigation.

$$y = \max(0, x) = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases} \tag{3.3}$$

Currently, the ReLU has become the most widely used activation function. First investigations of it were conducted by Jarrett *et al.* [51] and Nair and Hinton [52]. Its main advantages are that it does not saturate for positive values and that it is very easy to compute. Also, it has been shown that rectifying neurons are a better model of biological neurons and yield equal or better results than sigmoidal or hyperbolic tangent activation functions [53, 54].

Typically, all activation functions in a neural network are the same, except for the output layer, in order to keep the structure of the network as simple as possible. When choosing the activation function for the output layer, the nature of the data and the assumed distribution of target variables must be considered (cf. Bishop [55], p. 228). For example for standard regression problems, the activation function is typically linear. For classification tasks with more than two classes, the normalized exponential function (3.4) is applied. It is a multi-class generalization of the logistic sigmoid function (cf. Bishop [55], p. 198) and is also known as softmax function because it represents a smoothed version of the maximum function:

$$p(C_k|\mathbf{x}) = \frac{exp(a_k)}{\Sigma_j \; exp(a_j)} \quad . \tag{3.4}$$

It gives the classification probabilities p for every possible class C_k given an input vector $\mathbf{x}$. The a_j are the components of the input vector with $j = 1, ..., K$, where K is the number of classes.

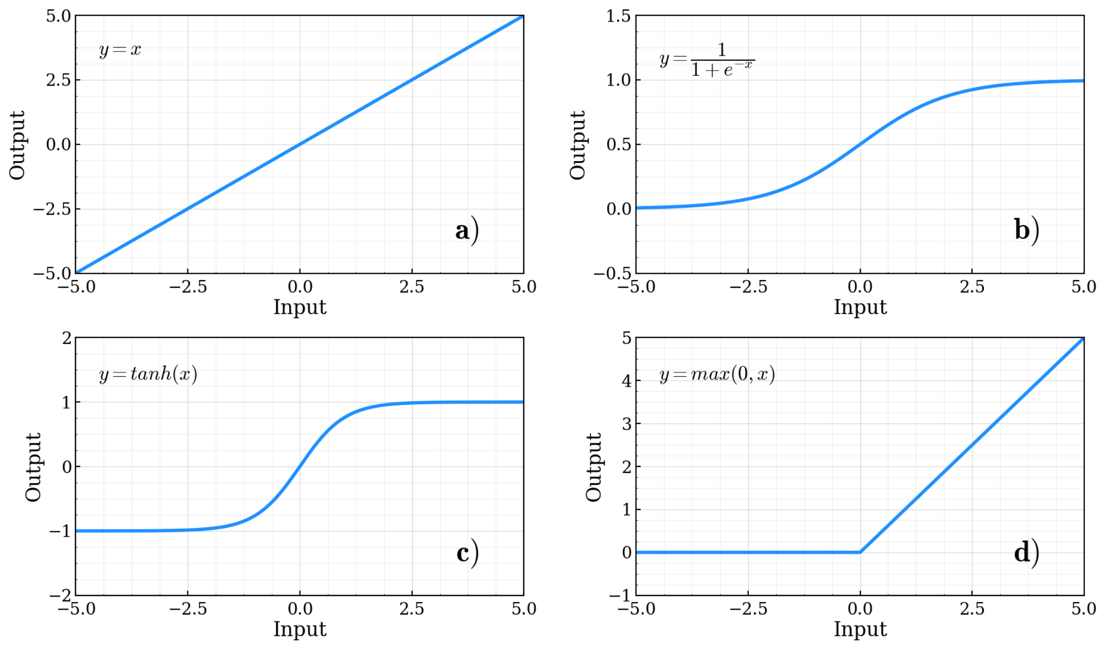


Figure 3.3: *Collocation of common activation functions that are regularly used in ANN applications. Panel a) represents the identity, which is an example for a linear activation function. This type is typically employed for output layers in regression tasks. In panels b) and c), the sigmoidal and hyperbolic tangent activation functions are given which historically have been the most popular nonlinearities applied in neural network training. Panel d) shows the ReLU function that has drawn increasing attention in recent years and exhibits superior efficiency in many cases [53, 54].*

3.6 Application of Neural Networks

Neural networks are a very powerful and versatile method and can be used to tackle almost any machine learning task. Theoretically, the so called universal approximation theorem implies that neural networks can potentially represent real-valued continuous functions arbitrarily well. There are various variants of this theorem. Two important ones will be briefly outlined in the following. Kurt Hornik found in 1991 that "standard multilayer feedforward networks with as few as a single hidden layer and arbitrary bounded and nonconstant activation function are universal approximators" ([56], p. 251). Similar results for networks with one hidden layer were already found in 1989 by Funahashi [57], Cybenko [58], and Hornik *et al.* [59]. However, it has been shown that networks with two or more hidden layers have some significant advantages regarding their approximation properties (cf. Pinkus [60], p. 182 ff.). Firstly, there is no theoretical lower bound on the error of approximation for ANNs with two or more hidden layers. Additionally, such networks allow for approximation by localized functions with compact support. Furthermore, Lippmann showed in 1987 that a neural network with one hidden layer is only capable of forming convex regions in the space spanned by the inputs, whereas a network with two hidden layers can form arbitrarily complex decision regions (cf. Lippmann [61], p. 16). Another variant of the universal approximation theorem deals with the type of activation function to be used. Leshno *et al.* established in 1993 that "A standard multilayer feedforward network with a locally bounded piecewise continuous activation function can approximate any continuous function to any degree of accuracy if and only if the network's activation function is not a polynomial" ([62] p. 861). The key point of the different variants on the universal approximation theorem is that multi-layer ANNs can approximate a wide array of functions if just minor conditions regarding the activation functions are met. Therefore, a great variety of machine learning tasks can be approached with this kind of setup.

In practice, ANNs are typically used to tackle two kinds of problem: regression tasks and classification tasks. In regression tasks, the outputs of the network represent continuous variables. In classification tasks, the inputs are to be matched with one out of several discrete classes, which represent nominal categories. In the present thesis, both kinds of problem will be relevant. For example, the classification of frames depending on the number of blisters visible is a classification task, whereas the localization or parameter estimation of individual blisters both constitute a regression task. For certain applications, special structures of neural networks have proven to be advantageous. For computer vision tasks for example, CNNs have become an established and widely used method. This kind of network will be used throughout the present thesis and will be discussed in Section 3.9.

3.7 Training of Neural Networks

The main point of machine learning techniques like ANNs is that decision rules do not have to be hand-coded but are learned automatically from training data. The resulting network ideally provides a model-based representation of the training data that relies on abstract features instead of idiosyncrasies of individual data points to make predictions. In the context of neural networks, the optimization of parameters is often called learning or training, respectively. In this section, the general procedure of training a neural network will be discussed and an outline of common implementation strategies will be given.

3.7.1 Setup of a Neural Network Before Training

In the case of supervised learning, training a neural network could also be conceived as fitting the overall network function to the training data by setting the adjustable parameters of the model so as to minimize a certain target function. The actual free, trainable parameters of a neural network consist of the aforementioned weight parameters as well as the biases for every single neuron in the network. Additionally, there are a lot of so called hyperparameters. These parameters are not altered during training but have to be set beforehand. Examples for hyperparameters would be the choice of activation functions for each neuron or the number and type of layers employed in the network. Hyperparameters greatly influence the performance of a network and their tuning is a key part of the development of deep learning applications. They will be discussed in Section 6.

The training of the weights and biases itself is usually conducted following the backpropagation algorithm (see Section 3.7.3) and typically consists of many iterations (cf. Section 3.7.4). Modern networks are very complex meaning that they can easily have several million parameters. Since there is no way of making reasonable assumptions as to which parameter values will yield the best results, they are initialized randomly before training. The main purpose of this randomization is to break the symmetry of the initial weights and thereby enable useful training. A standard initialization technique was introduced by Glorot and Bengio in 2010 [63]. This technique will be used for all weight initializations in the networks at hand. With respect to the biases, no theoretical reasoning as to which initialization to use is known to the author. Therefore, the default zero-initialization that is implemented in Tensorflow will be employed.

3.7.2 Loss Functions: Quantifying Network Performance

At the heart of the training procedure of a neural network stands the target function, which defines to what end the network parameters will be trained. In the context of neural networks, it is usually called loss function because it is a measure for the disparity between the prediction of the network and the desired outcome. It defines a distance between the predicted output vector, which is represented by the final layer of the network, and the target vector, which is given by the known outcomes (labels) of the training data. This quantity can be regarded as the loss or error of the network under the current weights. During training, the loss function is aimed to be minimized. The respective algorithms and optimization strategies will be discussed in the next three sections. The loss function directly determines the outcome of the training and therefore the performance of the network in general. It is therefore one of the most important components that one has to select prior to training. Typical loss functions include the standard mean-square-error (MSE) and the so called cross-entropy loss. The latter is typically employed together with the softmax activation function (3.4) in the last layer. The cross-entropy of two discrete probability distributions p and q with the same support χ is defined as

$$H(p,q) = -\sum_{x\in\chi} p(x)\ \log q(x)\quad . \tag{3.5}$$

The designation cross-entropy for this function was established by Good [64] and is widely accepted in the machine learning community. In the case of neural network training, p and q represent the true and the estimated distributions. Many algorithms require the training data to be one-hot encoded, which means that they only comprise a single true class. Such encoding conveniently allows for extra optimization during training. For classification problems, it has been shown that the cross-entropy loss function is a better choice than MSE because it leads to faster training and improved generalization [65]. In some cases, it is necessary to define a custom loss function that is specifically designed for a certain application. In general, every loss function can be written as

$$E(\mathbf{w}) = f(\mathbf{y}(\mathbf{x}_n, \mathbf{w}), \mathbf{t}_n)\quad . \tag{3.6}$$

The loss $E(\mathbf{w})$ is a function of the output vector $\mathbf{y}$ and the target vector $\mathbf{t}$. The output vector again depends on the input vector $\mathbf{x}$ and the network parameters $\mathbf{w}$. The input and output vector pairs are drawn from the set of known training data points $(\mathbf{x}_n, \mathbf{t}_n)$, with $n = 1, ..., N$. In the course of the present investigations, a custom loss function will be used. Details on this will be given in Section 6.5.

3.7.3 Error Backpropagation: Evaluating Gradients

Training a neural network means minimizing the value of the loss function given certain input and target vectors. Most training algorithms apply an iterative procedure of reducing the loss step by step. At every iteration, the process of decreasing the loss splits into two distinct parts. At a first stage, the derivatives of the loss function with respect to the parameters w_i are evaluated via the backpropagation algorithm. This will be discussed below. The second stage involves the adjustment of the parameters via certain optimization schemes like gradient descent. Those will be treated in the following two sections. The first step in the training process is the evaluation of the gradients. When there are multiple layers in the network, the chain rule must be applied to the derivatives of the loss function. In order to evaluate the gradients for all parameters in every layer of the network, the gradient information must seemingly be propagated backwards from the output layer through the network just like the information is passed forward through the network during inference. Rumelhart *et al.* established the term backpropagation for this procedure in 1985 and 1986, respectively [47, 66]. The algorithm yields the gradients of the target function with respect to all parameters in the network. It has superseded simpler methods such as the perceptron-convergence procedure [46] and has become the main algorithm for training neural networks today.

3.7.4 Gradient Descent: Updating Network Weights

The second stage in training a neural network is updating the network weights using the gradients of the loss function. This constitutes the actual learning process of the network. The weights are optimized so as to minimize the loss function. Hence, the task is to find the roots of the derivatives of the loss function in the multi-dimensional weight space. Due to the vast number of weights and the non-linearities introduced by the activation functions, there does not exist an analytic solution to this problem. Therefore, numerical methods must be applied. A classic root finding algorithm that is often used in machine learning is gradient descent. Its theoretical foundation is usually attributed to Cauchy [67]. However, he did not give a proof for the convergence of this method. This was rectified by Temple [68], who studied the convergence of relaxation methods of linear systems and by Curry [69], who gave a convergence proof of the gradient descent method for non-linear optimization problems. As in this paper by Curry, the gradient descent method is sometimes called steepest descent because the step is made in the direction of greatest decrease of the error function. Alternatives to the gradient descent algorithm for neural network applications will not be discussed here but are treated in depth in chapter seven of Bishop's book on neural networks [49].

The gradient descent algorithm utilizes the gradient information obtained with backpropagation so as to adjust the network parameters in such a way that the loss function moves closer to a minimum. Like other popular root finding algorithms, it takes an iterative approach to the problem:

1. Choose a random initialization of the weight vector $\mathbf{w}$.
2. Evaluate the gradients of the loss function $\nabla E(\mathbf{w})$ via backpropagation.
3. Take a step of a certain length, depending on the learning rate, in the unit $-\nabla E(\mathbf{w})$ direction. Then, repeat from step 2.
4. Advanced optimization schemes may include another step which involves the adaption of the learning rate before proceeding with the next iteration. This can happen according to some preset scheme or, in some cases, based on information that was gathered during past iterations.

The algorithm proceeds until one of the termination criteria is met. These could be: the predefined maximum number of iterations has been reached; the current point is close enough to a root; the gradient has become small enough so that further training is not considered worthwhile; no improvement was achieved for a certain amount of iterations. Whenever the algorithm is terminated before the maximum number of iterations has been reached, it is regarded as early stopping. This is a popular regularization technique that will be discussed in Section 3.8.3. The weight updates in step three of the algorithm are conducted according to (cf. Bishop [55], p. 240):

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \, \frac{\nabla E(\mathbf{w}^{(\tau)})}{|\nabla E(\mathbf{w}^{(\tau)})|} \tag{3.7}$$

where τ denotes the iteration index and η represents a quantity that regulates the step size that is taken into the opposite direction of the gradient. In the context of machine learning, the parameter η is called learning rate and is of major importance to the training procedure. It determines how fast a network learns in the course of a single iteration. The problem here is that if the learning rate is set too large, the algorithm overshoots and no minimum will be found. If it is set too small, a large number of iterations may be necessary to make significant progress towards the minimum, which can be very inefficient. Therefore, the challenge is to balance these conflicting demands. Since the optimal trade off is different for every network setup and every data set, it is desirable to have an adaptive learning rate that attunes to the respective environmental settings. This is implemented by advanced optimizers like ADAM [70]. For more details it is referred to the succeeding section.

When using gradient descent or any other optimizer, the following circumstance is to be considered. The ideal goal of the gradient descent algorithm is to find the global minimum of the loss function in weight space. Since the overall network function is nonlinear, the loss function will most likely be non-convex. This means that there can be multiple local minima. In particular, for symmetry reasons alone a large number of degeneracies will exist. Additionally, the huge number of dimensions will make it unlikely that the global minimum will be found within a reasonable number of iterations. Moreover, in practice it will typically remain unknown whether the minimum that was reached represents the global minimum. However, in general it is not necessary to find the global minimum. In most cases, finding a low enough point in weight space serves the purpose. Sometimes, it may be necessary to perform several executions of the optimization algorithm with different random initializations of the weights $\mathbf{w}$ in order to reach such a point.

3.7.5 ADAM: An Advanced Optimization Scheme

Gradient descent as described above is one of the simplest techniques to optimize the parameters of a neural network. Since its development, many variants and improvements to the method were published. Algorithms have been developed that are specifically adapted for the optimization of neural networks and that tackle specific issues that come along with this task. For example, standard gradient descent requires all samples in the data set to be regarded for each iteration. This is called batch learning. The evaluation of gradients for all data points simultaneously makes it less sensitive to idiosyncrasies of single data points and supports the learning of meaningful abstractions based on the entire data set. In contrast, online training approaches such as stochastic gradient descent (SGD) calculate the gradients for one data point at a time and update the network weights accordingly. This approach has been found to be superior to batch learning approaches when training neural networks on large data sets (cf. LeCun *et al.* [71], p. 546). Intermediate scenarios have been developed that try to harness the advantages of both online and batch learning approaches. They consider a small ensemble (mini-batch) of independent data points over which the gradients are averaged before they are used for updating the network parameters. Such mini-batch scenarios will be applied for the present network setups.

More recently, a new generation of optimization schemes has been established. Momentum strategies [72] like the Nesterov accelerated gradient descent [73] are used to reduce the variance in SGD and soften its convergence. They adapt the learning step so as to increase convergence speed in the relevant directions [74]. Furthermore, algorithms like Adagrad [75] and AdaDelta [76] adapt the learning rate for each parameter and for each optimization step individually, which again significantly increases the convergence speed. One of the most popular state of the art algorithms is the ADAM optimizer, which stands for adaptive moment estimation [70]. It utilizes the advantages of the aforementioned techniques and is likewise based on the usage of gradient updates involving exponential moving averages of squared past gradients [77]. ADAM has been improved by different researchers, for example through a "long-term memory" of past gradients [77]. Its main advantages are high speed, fast convergence, and the prevention of overshooting over minima, which can occur when the learning rate is set too large. A more detailed comparison of various gradient descent optimization algorithms can be found in literature [78].

3.7.6 General Remarks on Training Neural Networks

At this point, a general remark on the training procedure of neural networks is indicated. In general, any optimizer works best when the instances in the training data set are independent and identically distributed (cf. Geron [37], p. 416). In the case of online or mini-batch learning, it is important that the training data is shuffled beforehand or instances are drawn randomly from the data set. If the data set was sorted with respect to any characteristic, the network would simply consider this specific feature and would not build a meaningful and abstracted representation. This points to a general property of neural networks: they are very naive learners. They will always respond to the most primitive characteristic about the training data that enables a correct prediction for the training data set. Hence, the creation or selection of a balanced and comprehensive training data set is one of the most important and also challenging tasks when it comes to training neural networks. Another issue to regard when training neural networks is the variation in the training data. It should be as large as possible. The more diverse the data points are, the more complex the contour of the loss function will be in weight space. This will make it easier for the optimization algorithm to find meaningful improvements to its model so as to better predict the correct outcomes. This is because with a larger variance in the training data, even in adverse regions of weight space the negative gradient will point into a direction that improves the model at least to some extent. If the variation is low, on the other hand, the gradient might not provide any useful information for updating the weights in those regions.

3.8 Regularization: Ensuring Model Performance

This section will outline appropriate performance measures for neural network models and discuss typical issues like over-fitting that arise during training. Common regularization techniques, which are employed to address these problems, will be explained. A rigorous treatment of regularization in the context of computer vision has been performed by several researchers [79, 80].

3.8.1 Internal and External Validity: Measuring Performance

When training a neural network, one is generally interested in some kind of performance measure. It can be used to define a decision criterion for the abortion of the training process when the quality of the model suffices or for general comparisons between networks as well as benchmarking purposes. One could consider the value of the loss function in the current or final training iteration since it gives a quantitative measure of how well the model predictions match the true target values of the training data. This would be the internal validity of the model. However, the data that was employed during training cannot be used for evaluation because the model may start to remember the training data and will thereby likely predict the correct label. In this case, the model does not learn a meaningful and abstract representation of the relevant features but exploits idiosyncrasies of the training data instead. Hence, it will not generalize well to other data. This undesirable phenomenon is called over-fitting and will be discussed in the next section. It leads to a good internal validity measure although the model might not be applicable to new data. In practice, one wants to know how well a model performs on new data. This is quantified by external validity measures. Internal and external validity can deviate significantly. In order to determine the external validity, one can calculate the value of the loss function on a separate data set that was not used for training. Such a hold out validation set may be separated from the original data set for this purpose. Overall, the external validity represents an essential performance measure for neural networks. In order to increase the external validity and thereby ensure model performance, regularization techniques are employed that control the complexity of the model. Such techniques will be discussed in Section 3.8.3.

3.8.2 The Problem of Overfitting: Balancing Model Complexity

As mentioned before, ANNs are generally used to find model-based solutions to classification or regression problems. An important presupposition is that although the amount of data is large it can be explained in terms of a relatively simple model with a small number of parameters. When setting up a neural network, two generic problems that are closely related must be considered. If the model is too complex for the amount of information that is contained in the training data, it will likely overfit. This can even be the case for very large data sets since modern networks can have up to millions of adjustable parameters. When a model overfits, it means that it will adapt to the peculiarities of the training data and basically start to remember the data points instead of learn abstract rules about the structure of the data. In this case, the model may make perfect predictions for the training data but would not generalize well to new data. The external validity would then be low. On the other hand, when the number of adjustable parameters is too small, the model might not be able to represent enough relevant features of the data. In this case, the model lacks discriminatory capabilities and will not be able to perform the desired task. This issue is called underfitting and typically results in bad performance on the validation as well as the training set. In summary, over- and underfitting lead to an optimization problem with regards to the model complexity. It should be complex enough to catch all the variability in the data but also simple enough to not fit the model too closely to the particularities of the training set. Put differently, the model should have as few parameters as possible and as many parameters as necessary.

3.8.3 Main Regularization Techniques

To meet the aforementioned challenges of over- and underfitting, several regularization techniques may be applied. In general, ANN models rather tend to be too complex and the external validity is difficult to ensure. This is why most regularization techniques address the problem of overfitting and try to regularize the model complexity to that end. In this section, three commonly used regularization techniques will be presented that are employed for the models in the present thesis. Other techniques, for example Bayesian approaches for model comparison as described by von Toussaint *et al.* [81], will not be discussed here as they were not employed in any model.

Early Stopping

The training error of a neural network model is usually a non-increasing function of the iteration index of the optimization algorithm. This means that the internal validity of the model increases with every iteration step. However, this strict relation only holds for the training data itself. As outlined above, the internal and external validity of a model do not have to progress equally during training but can differ significantly. In practice, they usually improve simultaneously at the beginning of training but the external validity starts to deteriorate again in later stages of training while the internal validity still improves. This deviation between internal and external validity is a clear indication of nascent overfitting. One of the simplest regularization techniques utilizes the deviation of the external validity as a criterion for aborting the training process to prevent overfitting. It is called early stopping as it causes the optimization algorithm to not perform the predefined number of iterations but to terminate the training procedure early. The method can be provided with a patience parameter to let the algorithm run for a certain number of iterations without an improvement in external validity before early stopping is triggered. A prerequisite for early stopping is that a hold out validation set is available to compute the external validity after each iteration. In the context of training neural networks, these iterations of optimization algorithms like gradient descent are called epochs. An alternative and more straightforward way to retrieve the training status with the best performance is to store the model for every epoch. Then, the model with the best performance on the validation set can be selected after the training process is finished. This approach will be used in the optimization codes of the present thesis.

Amendment of Loss Function

A more involved technique to control the complexity of a model is to add a regularization term to the loss function which influences the resulting gradients and thereby the training outcome. This term is a function of the adjustable model parameters and can be used to specifically penalize larger or smaller weights. Two common variants of this method are L1 and L2 regularization. In L1 regularization, the penalty term is the sum of the norms of the network weights and hence targets smaller weights. L2 regularization, on the other hand, penalizes larger weights more by adding the sum of the squared norms of the network weights. It incentivizes weights to remain small and is therefore also known as weight decay. The penalty term δ for both L1 and L2 regularization is given by

$$\delta = \lambda \, |\mathbf{w}|^{\alpha} \quad . \tag{3.8}$$

In this formula, α determines the type of regularization, L1 or L2, and λ is called the regularization parameter. It determines how strongly the regularization affects the update of the weights $\mathbf{w}$. Such a regularization term can also help to remove undesirable degeneracies in weight space, which can for example occur when the softmax activation function (3.4) is employed (cf. Bishop [55], p. 236). It is also possible to apply a combination of both L1 and L2 regularization terms. A detailed introduction to the theoretical background of regularization, including more general formulas and some additional variants, is given in sections 5.4, 9.2, and 10.1 of Bishop's book on neural networks [49].

Dropout

Dropout is one of the most frequently used regularization techniques. It was first published by Hinton *et al.* in 2012 [82] and later improved upon by Srivastava *et al.* in 2014 [83]. The idea is to prevent neurons from co-adapting with their neighbours and solely relying on a few neurons in the previous layer. Instead, they are supposed to be trained partially independent from their neighbours. This forces the neurons to build more diverse connections and thereby become less sensitive to slight changes in their inputs. The implementation of dropout is straightforward: In whatever layer of the network it is applied, it causes some randomly chosen neurons to be excluded from training during the next iteration. This means that these units will have an activation level of zero for this particular iteration. The percentage of neurons that are inactivated is called the dropout rate p. It is a hyperparameter that has to be set prior to training. Dropout has been found to significantly improve the network's generalizability [82]. Therefore, it has been be applied in all models that were developed for the present thesis.

3.9 Convolutional Neural Networks

The branch of machine learning that deals with computer vision tasks has made enormous progress in recent years and receives ever more funding due to its anticipated applications in robotics, autonomous driving, and manufacturing processes. Today's software solutions for such tasks usually employ CNNs [42]. They are a special kind of ANN with a unique setup and versatile application [84]. The first neural network with a CNN-like architecture was the neocognitron proposed by Fukushima [85]. Earliest applications were made by LeCun *et al.* to the recognition of handwritten digits in ZIP codes [86] and later generalized to document recognition [87]. So far, they have proven to be very effective as well as efficient in solving problems involving the automated evaluation of image data. This section will outline their main components and algorithmic implementation as well as their general structure. Subsequently, some major advantages and disadvantages of their application will be discussed.

3.9.1 Working Principle

There are three distinct design elements about CNNs that make them very effective in image analysis. Two of them were initially inspired by the visual cortex of mammals, which was described for the first time by Hubel and Wiesel in 1959 [88]. To what degree the human perceptive processes are analogous to the one of CNNs has been investigated by Lindsay [35]. One of the main characteristic of the mode of operation of CNNs is its hierarchical structure of feature recognition [89]. Analogous to the human perceptive process, early layers in a CNN are sensitive to small and simple features like edges, while successive layers look for more and more complex patterns and combinations of smaller features, until eventually they identify entire objects (see for example Krizhevsky *et al.* [90]). Since this hierarchical structure of feature recognition is prevalent in many applications and early features are typically very similar, attempts are made to utilize pretrained layers for transfer learning [91]. Another design element of CNNs focuses on the spatial correlation of pixels in images. Specifically in image data, the correlation between pixels that are close to each other will be larger than between pixels that are further apart. This is utilized in the structure of CNNs via the concept of local receptive fields, again analogous to the human visual cortex. This means that neurons only have a very small region, i.e. perceptive field, from which they receive their input. Analogously, hidden units will only have connections to a fairly limited number of neurons in the previous layer which lie in the same region. An additional feature of CNNs exploits the fact that the appearance of objects or features is typically independent of their location in the scene. This translational invariance can be represented in the structure of a CNN by applying the same weights to all neurons in one layer, regardless of which region of the image their receptive field lies in. More precisely, the weights are shared only among neurons in the same feature map (cf. Section 3.9.2). This property is called weight sharing and allows to significantly reduce the number of free parameters in the network and therefore reduce its complexity and the computational resources required for its training. All three of these distinct design elements of CNNs are reflected in its setup by characteristic components. Local receptive fields and weight sharing are implemented via convolutional layers and hierarchical feature detection is enforced by the subsampling that happens in pooling layers. These components will be treated in the following.

3.9.2 Important Layers in CNNs

There are two distinctive structural components that characterize CNNs. One of them is the convolutional layer, in which the input or intermediate feature maps are convolved with kernels. These kernels represent features, which the network tries to recognize in the image. The other major feature of CNNs is the pooling layer, which subsamples the feature maps and enforces the learning of more abstract representations. Since these components are of major importance for the understanding of the methods applied in the present thesis, both of them will be explained in detail in the following sections.

Convolutional Layers

The convolutional layer is the core building block of a CNN and reason for its designation. Most of the computation and the actual feature recognition is happening in this type of layer. Convolutional layers apply a process that is equivalent to the mathematical method of convolution to the respective input image of the layer. Instead of the convolution of one function with another, however, the image is convolved with a filter. A filter is a multidimensional arrangement of numbers that represents certain features and is also called a kernel. The filter numbers are the weights between neurons of two successive layers. Typically, the filter is much smaller than the input image. It is shifted across the image and its numbers are multiplied by the input pixel values at the respective location in the image. This procedure is depicted in Fig. 3.4. The result of the convolution operation is the sum of these products at every location in the image. The better a region in the image matches the filter, the higher the value will be. Since a filter represents certain image features, the output of a convolutional layer is called a feature map because it highlights where in the image the feature of the filter is present. All units in the resulting feature map share the same weights with respect to the neurons in their individual local receptive field in the previous layer. This sharing of weights significantly reduces the degrees of freedom and has some favorable effects that will be discussed in Section 3.10. In each convolutional layer, several filters can be applied in order to detect different features resulting in multiple feature maps. In that case, the filters in the next layer will have a 3-D shape and their receptive field will span across all previous feature maps. Thereby, the successive layer can combine different features so as to compose more complex features and build abstract representations

In practice, the shape of the filters is typically quadratic and their size is typically between 3×3 and 7×7 pixels to keep the structure symmetric and the computational costs low. The small size of the filters means that neurons are only connected to a small number of units in the previous layer, which is characteristic for the aforementioned local receptive fields. From Fig. 3.5 it is apparent that the convolution with a filter reduces the size of the image across layers. If the size is supposed to be conserved, the input for each layer needs to be padded according to the size of the filter. A common padding type called "zero-padding" involves adding zeros around the image. Additionally, the step size for the shift of the filter during the convolution, which is called stride in the context of CNNs, can be adapted as well. The concrete choice of padding type and stride length are additional hyperparameters and will be discussed in Chapter 6.

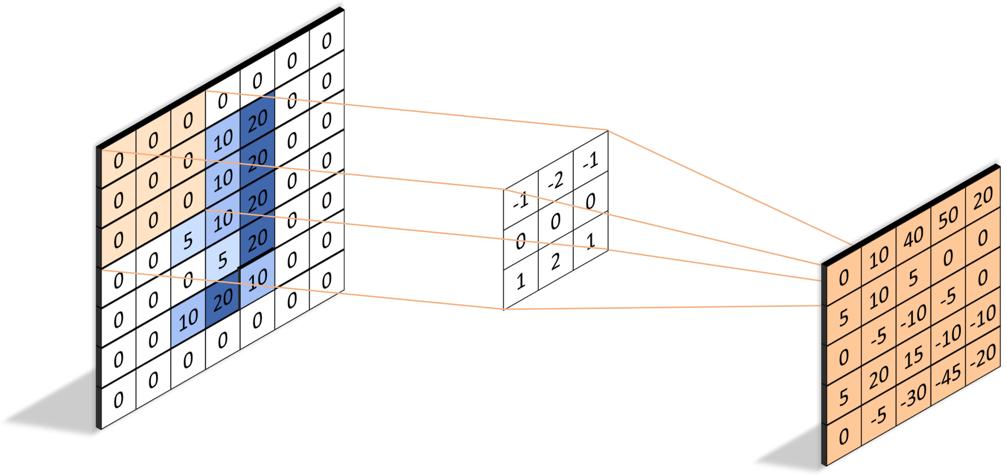


Figure 3.4: *Illustration of the convolution operation. The filter is moved across the input image and applied at each position. For every application, the sum of the products of the kernel values and the input unit activations represents the result of the operation. The entirety of all kernel applications makes up the resulting feature map. In this specific case, a horizontal Sobel filter is applied, which detects horizontal edges.*

Pooling Layers

In CNNs, pooling layers are often inserted to apply nonlinear sub-sampling to the feature maps of the previous convolutional layer. The purpose of this downsampling is to reduce the complexity of the model as well as the computational effort needed for its training and to lower the memory footprint. Pooling layers partition each feature map into subregions, which usually have a 2x2 shape and do not overlap. Then, a nonlinear pooling function is applied to each subregion separately. There is a large number of potential pooling functions, however just a few are regularly employed in practice. Historically, average pooling was the first widely used pooling function. It yields the average of the values in the respective subregion of the feature map. Nowadays, a different function called max pooling is applied, which tends to perform better [92]. In the case of max pooling, the result of the pooling operation is the maximum of the values in each subregion. Both average and max pooling are illustrated in Fig. 3.5.

Another major effect of pooling layers is that they increase the perceptive field of individual neurons from layer to layer and at the same time decreases the resolution of the image. Thereby, the learning of meaningful abstractions and combinations of features is explicitly enforced at every stage of the network. This facilitates the job of the network to proceed from the detection of simple features in the first layer to

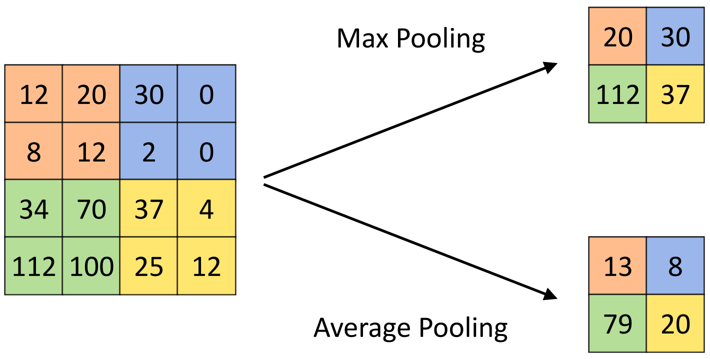


Figure 3.5: *Illustration of the two major variants of the pooling operation. The image or the feature map gets divided into subregions onto which the pooling is applied. Max pooling selects the maximum value of each subregion, whereas average pooling computes the average of all the values in every subregion.*

the identification of complex objects in the last layer. This progression is implicitly required through the structure of the CNN (cf. Hadji and Wildes [93], p. 10 f.).

3.9.3 Structure of a CNN

Just like standard ANNs, CNNs have a layered structure. A typical setup of a CNN is depicted in Fig. 3.6. It displays an exemplary network for the classification of handwritten digits. In general, the input layer of a CNN represents the digitized input image, potentially comprising several channels, e.g. for RGB colors. The input layer is typically followed by a set of alternating convolutional and pooling layers. As explained in the previous section, this arrangement leads to an increasing level of abstraction throughout the network. At the end of the network pipeline, a couple of fully connected layers are applied to the flattened 1-D representation of the feature maps. They perform high-level reasoning and the final assessment of the network. As outlined in Section 3.6, there should be at least two fully connected layers in the network for it to be a universal approximator. In the output layer, the final transfer to the output in the specified format takes place. This could be single or multiple regression outcomes as well as classification results. In the example in Fig. 3.6, the output comprises classification probabilities for all possible digits from one to nine.

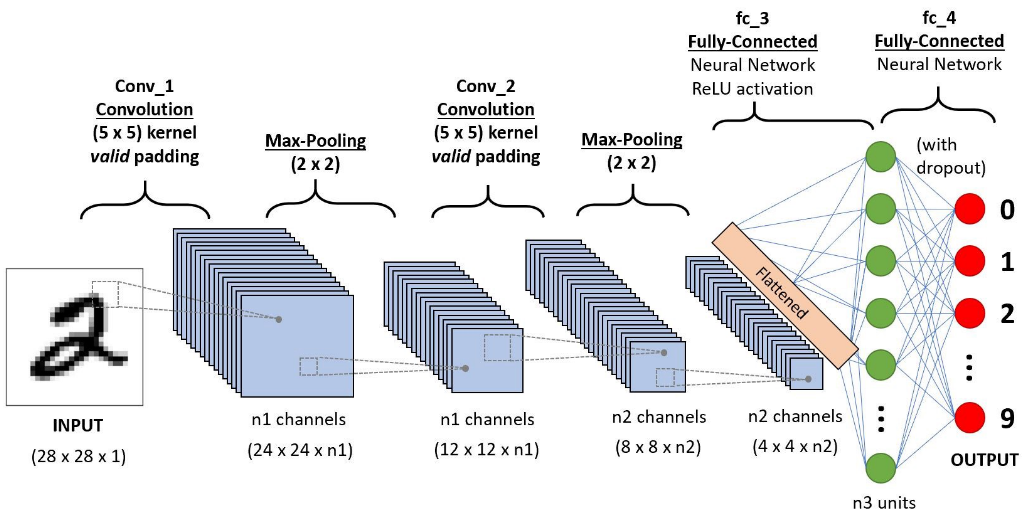


Figure 3.6: *Depiction of a typical CNN setup. Characteristic are the alternating convolutional and pooling layers in the first part of the network structure and the arrangement of several dense layers at the end right before the output layer. Image taken from [94].*

3.10 Applicability of CNNs for this Investigation

CNNs have become the state-of-the-art method for computer vision tasks [42] on the account of several features that make them favorable to other methods. Firstly, they exploit both the local spatial correlation of pixels and the translational invariance of the features by local perceptive fields and weight sharing, respectively (cf. Section 3.9.1) [89]. This is especially advantageous in object detection tasks because images can contain multiple instances of an object anywhere in the image. Due to these principles, the number of weights is much smaller than it would be in a fully connected network. The pooling layers reduce the amount of free parameters even further. However, despite the significantly smaller number of parameters, CNNs still perform very well on most tasks while being much more efficient. Additionally, the reduction of complexity helps to avoid overfitting (cf. Section 3.8.2). Another benefit of CNNs is their explicit enforcement of higher levels of abstraction at each layer by means of their structure and the application of pooling layers. This hierarchical feature detection is useful because features typically correlate and contribute to higher order feature detection mostly when they are about the same size or on the same abstraction level. The advantage is that the network does not need to infer these levels of abstraction implicitly in the course of training. Its structure facilitates this aspect independently from training. Finally, neural networks not only allow to actively train features that are relevant for the desired identification of objects but also allow for blacklisting. Thereby, unwanted features and disturbances of any kind can be explicitly 'untrained' by the network as long as appropriate training data is available.

On the other hand, CNNs also bring along some challenges. One of the major problems is that the interpretability of the acquired models is usually very limited, as is true for all neural network applications. It is very hard to find out what the network is actually doing at the intermediate layers. Often times, the understanding of the information processing remains low and the network rather operates like a black box. Hence, prior knowledge on how to design the network is usually scarce. Therefore, the process of training a neural network and tuning the hyperparameters involves a significant amount of trial and error iterations. However, some attempts were made to better understand neural networks [95]. Specifically for CNNs, a variety of techniques has been developed to interpret the computations and learned parameters at intermediate layers [96, 97]. For example, deconvolution approaches are used to visualize features that the network responds to [98], while other studies investigate the differences of recognition processes between humans and deep neural networks [33, 34].

Overall, CNNs represent a powerful tool that is capable of solving a great variety of problems. Specifically, they are well suited for addressing the regression and classification tasks that are part of the blister investigations in the present thesis. The setup of the specific CNNs employed in the present investigations will be discussed in detail in Chapter 6.

Chapter 4

Experimental Data and Preprocessing

The main goal of this thesis is to achieve the large-scale and time-resolved study of blistering on plasma-exposed metal surfaces. For this specific investigation, the blistering on a molybdenum sample was considered since in this case the blister features become quite large. This allows to conduct the development of the methodology and analysis framework without being restricted by limits that are imposed by the experimental setup. However, the framework developed here can in principle be transferred to other materials as well, for example tungsten. For the present investigation, the specimen was exposed to a deuterium plasma for some 25 hours. The details of the experimental setup will be discussed in Section 4.1.1. The experiment yields in-situ images of the sample during the plasma exposure. After the acquisition, the images have to be subjected to some preprocessing steps to prepare them for the automated analysis with convolutional neural networks later on. These steps include the alignment of consecutive images, the application of a background subtraction, and denoising. They will be discussed in Section 4.2. The sample was observed during the entire experiment and the image quality resolutes all major blister features. By means of the preprocessing techniques, important features in the images can be highlighted and undesirable effects can be reduced. Individual blisters can clearly be identified as well as characterized. Overall, the employed experiment and the applied methods yield appropriate image data that can be further evaluated with automated routines.

4.1 Experimental Setup and Image Acquisition

For the investigation of blistering on metal surfaces, an in-situ experimental setup was used. It allows to study the temporal evolution of blisters statistically, as well as the development of individual blisters. For the loading of the samples with deuterium, the plasma experiment PlaQ was used. The details of this experiment and the specific settings for the present investigation will be discussed in Section 4.1.1. During the loading of the sample with deuterium, the surface was observed with a camera. Its high resolution enables the identification of individual blisters. The specification of the camera and the mode of operation for the image acquisition are outlined in Section 4.1.2.

4.1.1 PlaQ: In-situ Observation of Blistering on Metal Surfaces

For the purpose of the present investigation, an experimental setup is required that allows the live observation of blistering on metal surfaces. Such a configuration is provided by the PlaQ experiment. It is a plasma experiment that was developed at the Max-Planck Institute for Plasma Physics in Garching [6]. The sample considered for this thesis was exposed to a deuterium plasma from this source. At this point, I will only give a brief review of the general setup as well as some key features that will be relevant during this investigation. For a more detailed description of the device and important parameters it is referred to the original paper. A schematic representation of the experimental setup is depicted in Fig. 4.1.

In the PlaQ experiment, the plasma is ignited and maintained by electron cyclotron resonance (ECR) heating, which is implemented by a 2.45 GHz magnetron with P_{max} = 1200 W. The plasma is operated at low temperatures as well as low pressures (0.1 - 1.0 Pa) and uses D_2 as the operating gas. The main discharge is confined to a cylindrical steel mesh cage and therefore spatially separated from the sample holder. Through an aperture in the bottom steel plate of the cage, the plasma beam can reach the specimen. The beam can be blocked with a shutter underneath the aperture. By this means, the exposure of the specimens to the ion beam can be precisely controlled. The sample holder itself is electrically isolated from the vacuum chamber and can be biased up to -600 V. This allows to adjust the impact energy of the ions on the specimen without strongly influencing the main plasma. The irradiation of the sample holder by the ion beam is approximately homogeneous. The deuterium ion flux density has only a weak dependence on the bias voltage and has an absolute value of about 10^{20} $\mathrm{Dm^{-2}s^{-1}}$. The ion flux is constant even over long periods (up to 12 days). Thus, large deuteron fluences can be achieved. The deuteron flux is mostly carried by D_3^+ ions, while D^+ and D_2^+ contribute only small fractions. Hence, the typical energy per deuteron is one third of the total ion energy. An important feature of the setup are its multiple windows on opposing sides of the chamber. They allow the illumination of the sample with a spotlight and the simultaneous observation of the specimen with a camera from the outside.

For the investigations at hand, a molybdenum (Mo) sample was exposed to a deuterium plasma for about 25 h and 35 min. The sample was held at a temperature

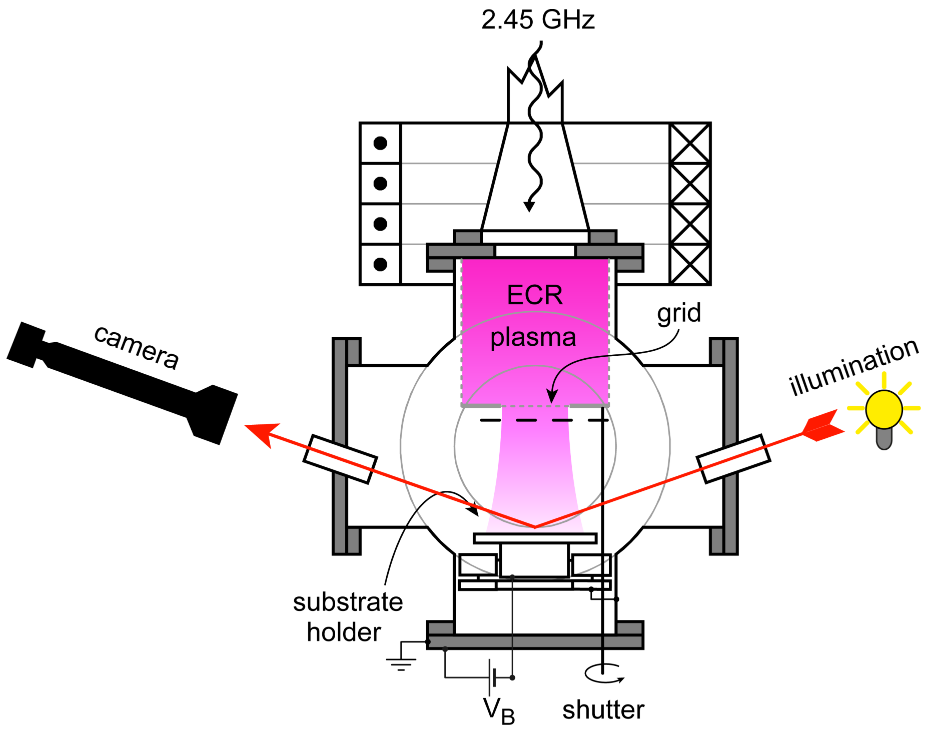


Figure 4.1: *Scheme of the experiment 'PlaQ'. The main deuterium plasma is confined to a stainless steel cage while a plasma beam can pass through an aperture and reach the specimen. The sample holder is observed through a window by a camera with a telecentric lens and illuminated by a light source which is placed in line of sight of the camera on the opposing side of the setup. The figure is based on the illustration in the original paper [6].*

of 370 K and the sample holder was biased to 100 V. This resulted in an energy per deuteron of 38 eV (cf. the original PlaQ paper [6]). The flux was 10^{20} $\mathrm{Dm^{-2}s^{-1}}$. The specimen was mechanically polished and had dimensions of 12×15 mm. A marker was carved into its surface to facilitate the alignment of the images. This marker had an approximate size of 3×3 mm and can be seen in Fig. 4.2.

4.1.2 Image Acquisition

For the in-situ observation of blistering on the specimens, two windows on opposing sides of the sample holder are available in PlaQ. In front of one of them, an adjustable LED light source with up to 75 W and a color temperature of 6500 K is placed. The spotlight is directed at the sample holder and is used to illuminate the specimen. Before the other window, a camera is placed at a distance of 365 mm and at an angle of about 20 degrees, which is the same as for the light source. Consider Fig. 4.1 for

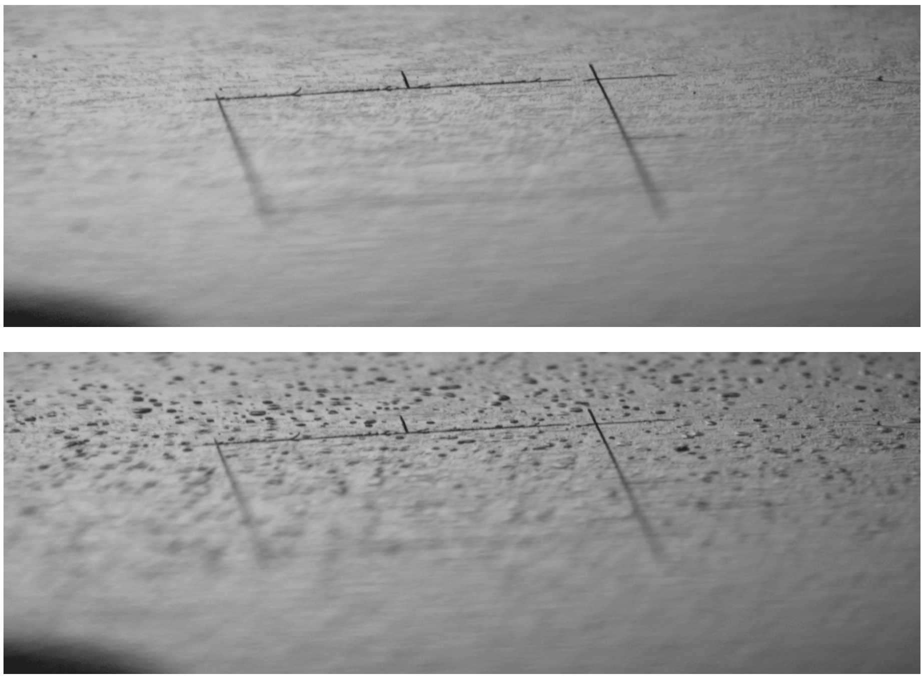


Figure 4.2: *Sample images of a molybdenum specimen that were acquired in-situ with the PlaQ experiment. At the top, the sample has not yet been exposed to any plasma yet. At the bottom, the sample has been exposed to the deuterium plasma for approximately 500 minutes. Note that the employed lens is telecentric, which means that there is no perspective in the images.*

an illustration of the general setup. The camera has a magnification factor of 1,870, a depth of focus of 2 mm, and a working distance of 365 mm.[1] It employs a telecentric lens which means that there is no perspective distortion in the acquired images and the magnification is independent of distance. Hence, objects appear the same size regardless of their position on the specimen. Thereby the automated evaluation of the images at a later stage is facilitated.

The optical resolution of σ of the acquisition system can be computed according to

$$\sigma = \frac{\lambda}{2NA} \quad , \tag{4.1}$$

where $\lambda = 500$ nm is the wave length of the light and $NA = 0.055$ is the numerical aperture. The optical resolution is hence 5 μm, which is equivalent to 3 pixels. Under the present circumstances, blisters on Mo have diameters of 50 to 500 μm and heights of 5 to 15 μm. The gradient angles of the blisters, i.e. the slope of the side walls, reach up to 15 degrees. This means that the blisters get illuminated well by the light source and that they can be observed entirely and with high resolution by the camera.

[1]A detailed data sheet of the employed camera system is provided in Appendix A.

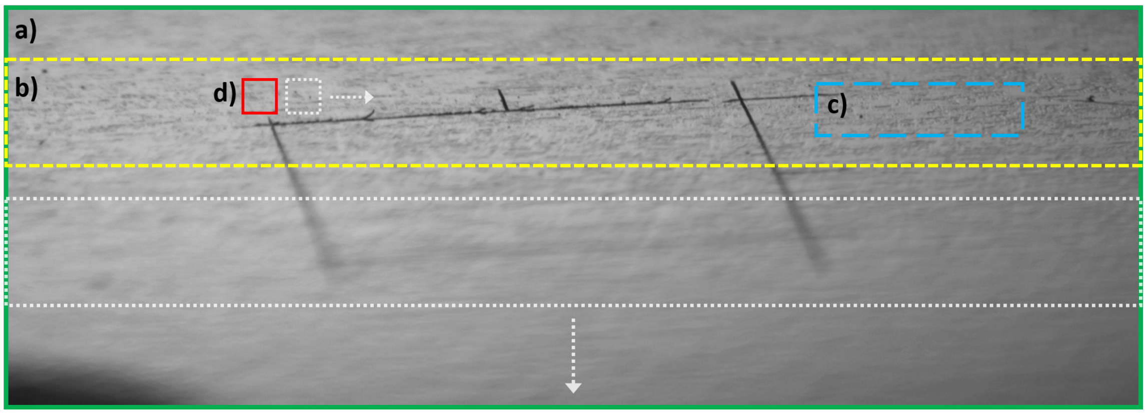


Figure 4.3: *Overview of the terminology regarding different image segments used in the present thesis. Panel a) denotes the complete frame, panel b) is an image slice, panel c) is an image section, and panel d) is called snippet and will be used as input to the CNNs. The shift of b) for the acquisition of the image slices during a full image cycle and of d) for the sliding window scheme are indicated by boxes and arrows with white dotted lines. The aggregate of all 29 image slices constitutes a single image cycle and covers the entire frame.*

Due to the limited depth of focus, the camera is shifted back and forth to cover the entire field of view. To this end, it is mounted on a motorized optical rail. A scan across the entire specimen, called a cycle, involves 29 individual image acquisitions, which are called slices. An overview of the terminology of different image segments is given in Fig. 4.3. The acquisition of all 29 slices in a single cycle was set to take about 5 minutes and subsequent slices have significant overlap of the region where the image is well-focused. In total, 308 cycles of image acquisition were conducted during the experiment, out of which the first 100 will be used for the analysis in Section 8. Each image has a size of 4096×1440 pixels and a gray value resolution that is equivalent to 12 bits.

In the present thesis, it will frequently be referred to all sorts of different image segments from the original image data. Therefore, the respective terminology will be briefly defined here. An illustration of all the different segments is given in Fig. 4.3. In the figure, a) denotes the entire original frame, which is identical to the image in Fig. 4.2. Further, b) represents one out of the 29 image slices that collectively cover the entire frame and make up a single image cycle. The shift and the acquisition of the other image slices is indicated by the large box and the arrow with the white dotted line. During the discussion of the images and the analysis results throughout the present work it will often be referred to various smaller image sections to highlight specific features. These image sections are exemplarily represented in Fig. 4.3 by c). Finally, the 64×64 pixel snippet that will be extracted from the image during the application of the sliding window approach (cf. Section 6.1) to feed it to the CNNs as input is illustrated somewhat magnified by the small red box and labeled with d). The shift of the snippet during the sliding window procedure is again insinuated by a box and an arrow with white dotted lines. The reader is made aware of the fact that both the successive image slices and the successive snippets actually feature significant overlap and were only indicated separately in the figure to make it more clear.

4.2 Image Preprocessing

Before the images can be evaluated with CNNs, they have to undergo several preprocessing steps in order to deal with a couple of issues. For example, consecutive frames are randomly shifted due to vibrations of the setup. This is addressed with image alignment, which ensures the persistence of object parameters like its position across time. It also lays the basis for the subsequent background subtraction. The texture and the irregularities on the surface of the specimen constitute another issue. They require a background subtraction which allows to highlight the changes in images over time and therefore increase the visibility of blisters. Also, some noise is present in the images which is rectified by advanced denoising techniques. All of these preprocessing steps will be explained in more detail below. Overall, the employed techniques significantly reduce the abundance of unwanted features in the images while increasing the visibility of important blister features and the persistence of parameters across time. Thereby, they contribute greatly to the success of the present investigations.

4.2.1 Image Registration with Hugin

During the acquisition of the images as described in the previous section, the whole experimental setup experiences some movement due to unavoidable vibrations of experimental setup components and external factors. These disturbances become visible as minor and major shifts in the images. These shifts are not desirable and need to be corrected for in order to make the locations of objects on the specimen surface consistent across time. Additionally, a good alignment allows for a background subtraction (see Section 4.2.2) which in turn simplifies and enhances the quality of the automated detection of blisters. It therefore constitutes a major success factor for the research goals of this thesis.

For the alignment of the images, the open source panorama photo stitching program Hugin was used.[2] It allows to automate most of the necessary processing steps and is

[2]The Hugin software was used in the 2019.2.0 version for 64 bit Windows operating systems. It can be downloaded from the Hugin webpage [99].

a powerful tool that yields very precise results. For detailed information on specific features, one may consult the instructive tutorials available online [100] or the online source code documentation [101]. The concrete choice of settings and processing steps widely follows the instructions from a publicly accessible application example [102]. It may serve the reader as a detailed guideline. For the study at hand, a single image slice over the course of 100 image cycles was considered. Hugin considers every pair of images from the complete time series separately. This results in a total of 4950 pairings. Hugin then automatically searches for control points in every image pair individually. These control points are meant to be idiosyncratic and ideally unique to the image. Thereby they can be recognized in the other image and hence allow for an alignment of both images. The number of identified control points varies but is typically in the range of 2200. If required, individual control points can be edited by the user. For example, the control points with the smallest coincidence may be excluded from further consideration since they might be misidentifications. In a second step, Hugin tries to align all images at the same time so as to minimize the deviation of all control points. Multiple iterations of this control point optimization may be applied in order to improve its accuracy. Prior to the alignment, the user can choose which alterations in the images are to be considered, for example translations of the camera, change in viewing angle, or relative image positions. For the present study, only the position was regarded as an adaptable parameter. Subsequent to the alignment, the images are cropped to a region common to all images to allow for background subtraction and consistent tracking of features over the entire time period. This procedure is feasible because only minor shifts of less than 70 pixels occurred between images compared to an image size of about 500×4000 pixels. Additionally, the images in an acquisition cycle have significant overlap. Therefore, the information loss due to cropping is negligible.

4.2.2 Background Subtraction and Denoising

In order to reduce the intensity of texture and surface features as well as reflection and shadowing effects, a background subtraction is conducted. Hence, an appropriate background needs to be determined. To this end, the acquisition of the images as presented in Section 4.1.2 was started about 90 minutes prior to the plasma exposure. Hence, some 18 image cycles were completed before the plasma exposure began. During this time, only the original surface without any alterations has been observed. Therefore, these early images can be used to define an appropriate background. However, not all of them are aligned equally well and so the five images with the best alignment were selected, i.e. images from cycles 14 - 18.[3] For the implementation of the background subtraction, the selected images are averaged in order to make the

[3]Best alignment meaning with respect to the succeeding images that were used for further analysis, i.e. cycles 19-100.

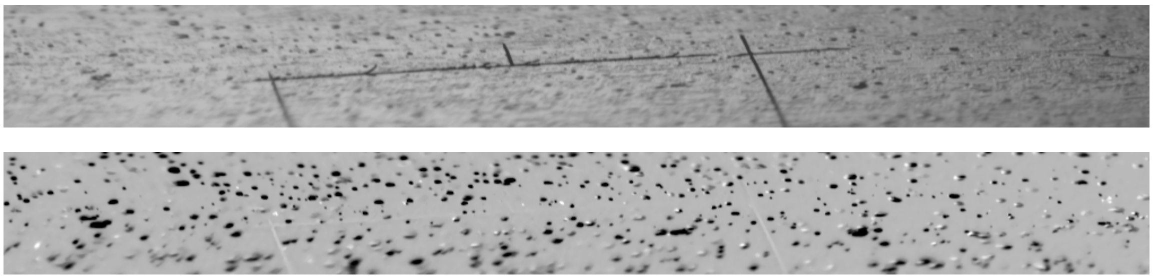


Figure 4.4: *Depiction of the background subtraction. At the top, an example of an image from the sample after the alignment and cropping is shown. Many blisters have already appeared. At the bottom, the same image is shown with the background subtraction applied. One can see that the background subtraction works very well. The marker on the sample is barely visible while blisters are highlighted nicely and exhibit clear contours.*

background more robust to noise, disturbances like dust particles, and artefacts in the images. The averaging also helps to reduce the effects of bad alignment between pairs of images from the background set and the rest of the image sequence. Finally, the background is subtracted from all other images. Afterwards, the images are rescaled to 8 bit grayscale 'png' format according to

$$X[i] = \frac{X[i] - X_{min}}{X_{max} - X_{min}} \times 255 \quad . \tag{4.2}$$

In this equation, X represents the entire image stack, $X[i]$ is one single image, and X_{min} and X_{max} are the minimum and maximum values of all pixels in X.

After the application of the background subtraction, an additional processing step needs to be performed to deal with the noise in the images. Naturally, there is some noise that originates from the acquisition process during the experiment. Also, some additional noise was introduced by the preprocessing steps described above. In order to diminish the noise, a special denoising technique was employed. It uses similar subregions in an image to compute their average and hence is called non-local means denoising. For details on this technique it is referred to the orignial paper by Buades *et al.* [103]. For the implementation in Python, a routine from the OpenCV library was used.[4]

The result of both the background subtraction and the denoising is depicted in Fig. 4.4. It is apparent that the two preprocessing techniques work very well. Virtually all of the texture is gone and even the very pronounced marker is barely visible. At the same time, blisters have become nicely highlighted and exhibit sharp contours, whereby they are easy to identify. In addition, the noise has been largely

[4]OpenCV is an open source library for computer vision applications. It was used in its 4.2.0.34 version for Python, which is available online [104].

removed from the image. It should be noted that blisters are typically darker than the surface and therefore become visible as dark spots in the background-subtracted images. This is also true for disturbances like dust particles, which may also occur. However, this happens rarely and does not cause too many misidentifications by the CNNs. Overall, the preprocessing steps described here and in the preceding section work very well and greatly facilitate the automated detection of blisters.

Chapter 5

Generation of Artificial Training Data

For the automated analysis of the image data, convolutional neural networks are used (cf. Section 3.9). In order to perform their task, these networks must first be trained with appropriate training data. The investigations at hand require a supervised learning setting which means that the true parameters (labels) of the training data are known and can be used to optimize the network parameters. One way to acquire the necessary labeled training data is to take real images and label them by hand. This is both laborious and deficient. An alternative way is to artificially create image data that resembles the real images. The advantage of the latter approach is that once such an automated procedure has been developed, the ground truth of the data is fully known and an arbitrary number of training instances can be generated as and when required. Ideally, the generation of artificial training data is completely automated with scripts and software applications. An additional advantage of such an automatized implementation is that new training data can be easily expanded or adapted and then produced at low cost. This is helpful when the required specifications have changed, a different sample is considered which looks different, or in cases where the preprocessing did not work as well.

For the training of CNNs with artificially created data, it is important that the images represent the relevant features of the real images and the objects that are to be identified as closely as possible. They have to reproduce all significant characteristics and should not exhibit any peculiarity that distinguishes it from real data. Otherwise, training will fail, as CNNs are very sensitive to slight changes and subtle characteristics in the training data. As outlined in Chapter 3, they adapt exclusively to the feature space that is covered by the training data and usually fail to transfer well to data that lies outside of this parametric environment.

For the applications at hand, the artificial training data was created with the ray-tracing software POV-Ray, which will be discussed in Section 5.1. The actual configuration of realistic blister scenes and their implementation in Python as well as POV-Ray are outlined in Section 5.2. For the verification of the artificially generated images, a number of different means are employed, which will be presented in Section 5.3.

5.1 Ray-Tracing Software POV-Ray

As outlined above, the artificial training data of CNNs must be as realistic as possible. With the open source ray-tracing software POV-Ray it is possible to create photo-realistic images from scratch for a large variety of settings. The software is used in its 3.7.0.0 version for Windows that was published under the AGPL3 License and is available online [105]. It was initially released in 1991 and has since been improved a lot and found a wide range of applications. Its documentation can be found online as well [106]. In general, POV-Ray allows to render fixed scenes with basic features like different cameras, light sources, objects, and ambient settings. In addition, it provides packages for rendering objects of various types. Among others, boxes and ellipsoids can be created and many of their properties can be adapted to specific needs. The adaptable parameters include color, shape parameters like height, width, and radius, as well as surface modifiers like texture, bumpiness, reflection, and more. In particular, metal surfaces can be imitated easily with pre-defined routines. Specific settings and the setup of concrete scenes will be discussed in the following section.

5.2 Configuration of Realistic Blister Scenes

The purpose of this section is to illustrate how photo-realistic blister scenes can be generated with POV-Ray. As a reference for both general scenic settings as well as specific blister parameter ranges, the Mo sample described in Chapter 4 after a plasma exposure of about 9.6×10^{23} Dm^{-2} was used. For reasons outlined in Section 6.1, only small snippets of the entire image slice of the real images are fed to the CNNs one by one. They comprise 64 x 64 pixels and cover an area of 217 x 638 μm^2 on the sample. In an area of this size, typically not more than four blisters will occur for the considered amount of fluence. Accordingly, the artificial images are rendered with a resolution of 64 x 64 pixels and only display small snippets with up to four blisters (see Fig. 5.1). The distance relations of all entities in the synthetic scene are proportional to their relative positions in the actual experimental setup (cf. Section 4.1). A camera with a telecentric lens and a point like light source are placed in the POV-Ray scene at the appropriate distances and angles. The basic setup consists of an object of type box that is placed it in the middle of the scene. It fills the entire image so that its surface resembles a plane. The viewing angle of the camera is chosen so as to exactly comprise a fraction of the surface that corresponds to the respective area on the real specimen. These settings ensure that the scene features the same mapping as the real experimental setup where 1.84 micrometer are enlarged and mapped onto a single pixel, which has a width of 3.45 micrometer. In POV-Ray, the quantities were chosen such that one spatial unit represents one micrometer. Blisters are emulated by distorted spheres that are partly absorbed into the plane. Their dimensions and depth under the surface are chosen such that the part that is visible above the surface looks like a blister.

As discussed in Section 3.7.6, the artificial images must reproduce as much variation as possible from the real images. To this end, several settings were randomized within certain parameter ranges. These specifications were implemented in Python and then formatted so as to serve POV-Ray as an executable input. Both the Python and POV-Ray scripts are provided in Appendix B along with the POV-Ray configuration file. They can be consulted for any details on the implementation of the generation of the artificial images. Therefore, only a brief outline of the parameters will be given here. Notably, only parameters that were adapted to create variation in the data will be discussed in detail. This is because they are of greater importance for ensuring the correspondence of the variation between the artificial and the real data. Their choice involved the detailed investigation of relevant variations in the real image data and according derivation of appropriate parameter ranges. The rest of the POV-Ray parameters, which were not varied, were mostly tweaked via trial and error and kept fixed afterwards. They were iteratively optimized by visually comparing the results with real images and adapting them accordingly. They are also very important for the realistic look of the images, especially with regards to the surface looks of the plane and the spheres, but do not contribute to the variation in the data. However, they were specifically optimized for the Mo sample and hence, if the specimen or the preprocessing steps are altered, these parameters must be adapted. Their specific settings can be gathered from the code, which is provided in Appendix B. The two most important static features that are employed to imitate the real surface looks of the Mo sample are a metallic reflection property of the surface plane and a marble texture, which can be adapted such that it resembles the texture on the Mo sample quite well.

Apart from the static features, a number of parameters were varied randomly in order to create adequate variation in the training data. For each parameter, a minimum value and a maximum value were defined. These values set the range out of which a random number is selected. In some cases, the randomized selection was additionally biased to better represent the parametric variation in the real data. An overview of these parameters is given in Tab. 5.1. It indicates the respective parameter range and gives some additional notes if applicable. One part of these dynamic parameters concerns the scenic surroundings. Here, for example the position of the light source was shifted randomly in both the vertical and horizontal direction so as to imitate different illuminations on the sample in the real experiment. These are

Randomized Parameter	Min. Value	Max. Value	Note
Scenic Environment:			
Light source lateral range	0	100,000	At 270000 distance
Light source vertical range	82,800	93,230	At 270000 distance
Plane texture randomizer	0	100,000	Random shift
Blisters:			
Initial Radius (R)	15	40	Bias to smaller values
x position	0	217	
y position	0	638	
Scaling of x-dimension	1	2.5	Bias to smaller values
Scaling of y-dimension	1	2.5	Bias to smaller values
Depth below surface	0.5 * R	0.93 * R	Bias to larger values
Rotation angle (degrees)	0	90	
Number of blisters	1	4	

Table 5.1: *Overview over the parameters that were varied in POV-Ray to make the variation in the training data more realistic. The first subgroup describes the parameters that determine the scenic environment. The second subgroup outlines all the parameters that affect the blisters. For each parameter, the respective range and some applicable notes are given.*

caused by the surface profile and curvature of the sample, which slightly change the illumination angle. In addition, the surface of the box was adapted by applying a randomized shift to the texture parameter.

However, the most important variation of the images happened with regards to the blisters. In the setup at hand, the parameters of every blister were randomized individually and are largely independent of the other blisters in the image. First of all, a sphere with a random initial radius between 15 and 40 μm was created. Subsequently, this sphere was deformed in both x and y directions independently, resulting in a randomized ellipsoidal shape. Afterwards, this object was randomly placed on the surface with the majority of its volume below the surface plane. The x and y location parameters range from one end of the visible section to the other. This means that the mid point of all blisters lies within the visible section of the scene. It is noteworthy that both the distortion and the overlay of the surface over the ellipsoid alter the apparent size of the blister, which hence does not correspond to the initial radius. The true size of the blister is eventually defined as the area of the blister as if viewed from the top and measured in pixels as given by

$$a = \pi x_{min} x_{maj} \times 2.94 \quad . \tag{5.1}$$

Here, a is the area of the blister, x_{min} and x_{maj} are the minor and major axis of the ellipse that is constituted by the intersection of the surface plane with the previously created ellipsoid, and 2.94 is the correction factor for the angle of view at which the scene is observed. For details on this, it is referred to the respective code listing in Appendix B. As a final step, a random rotation is applied to the blister-like object. Overall, the described settings result in the placement of an ellipsoidal section on the surface that features random dimensions, shoulder angle, height, and rotation. The above process is repeated between 0 and 4 times for each image, according to the requirements of the respective CNN scenario (cf. Section 6.1). The reader is pointed to the fact that it is possible for blisters in the POV-Ray setup to be placed inside one another, since the locations are chosen by randomized routines. This is avoided by keeping track of all blister positions and prohibiting the choice of a nearby location for the placement of a new blister. Furthermore, for some scenarios it was necessary to make additional adaptations to the settings, which will be discussed in Section 6.1.

After the setup of the scene parameters in Python, POV-Ray is executed with these specific settings and produces a 64 x 64 pixel 8 bit grayscale image in png format. For each required CNN scenario, a total of 17,000 images are produced with the exact same randomization parameters. The concrete values of the parameters for each image instance are saved to a file. However, only target parameters that are aimed to be determined by the CNNs later on are included in the output file. These include the number of blisters in the image as well as the location and the size of all blisters. Other parameters are not of further interest and are hence disregarded. At the end of the procedure, this file contains the true parametric labels for each image. These will be necessary for the supervised training of the CNNs. After the creation of the artificial images, a final postprocessing step is necessary. The resolution of the camera in the actual experimental setup does not correspond to one single pixel but is equivalent to 2.76 pixels. Therefore, an intentional blur needs to be applied to the artificial images to make them look as realistic as possible. This is implemented with a Gaussian blur that has a standard deviation of 2.76.

The overall results of the above described pipeline for the generation of artificial training images are very good. Some examples of such images can be seen in Fig. 5.1, where some artificial images (bottom) are compared to some real images (top). As one can see, the artificial images are almost indistinguishable from the real images even for the human eye. This makes it likely that the training of the CNNs with the generated images will be successful. However, to confirm this initial assessment, a more detailed analysis of the validity of the artificial data is necessary. This will be discussed in the next section.

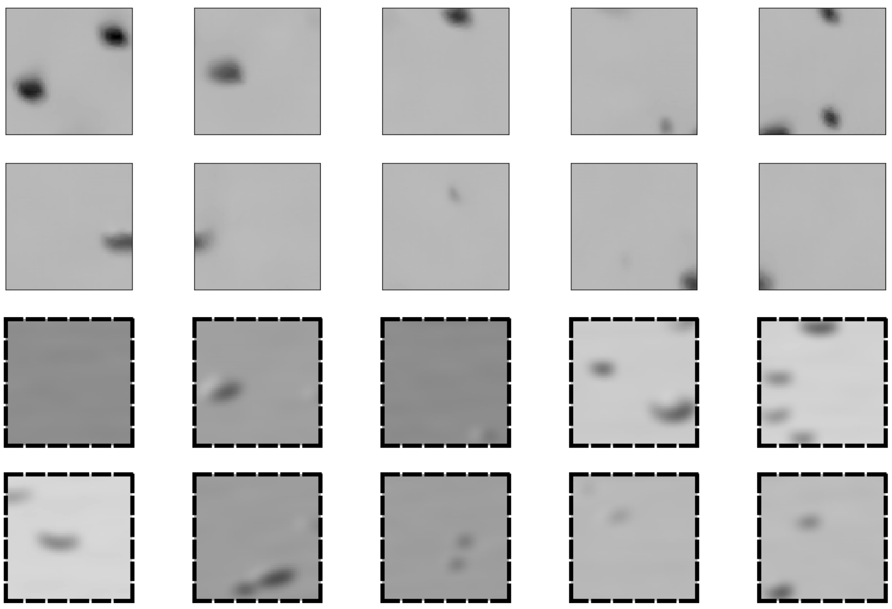


Figure 5.1: *Visual comparison between real images (top rows) and artificially generated images (bottom rows), the latter of which are indicated by black dashed lines around them. This comparison shows that the images created with POV-Ray correspond very well to the real measured images and in particular all major blister features are resembled to a great degree. There are only minor systematic differences discernible between the images, for example blisters in the real images seem to be somewhat darker.*

5.3 Validation of Artificial Images

The purpose of this section is to verify the accordance of the artificially generated images with the real images. To this end, a number of means were employed that allow for a detailed comparison between artificial and real images. First of all, an initial visual analysis was conducted where artificial and real images were compared by eye. This is a valid and important indicator of the validity of the synthetic data since the human eye is very good at perceiving differences in images. If a human does not notice significant differences, a CNN is likely to deem them similar as well. Such a visual comparison can be seen in Fig. 5.1. In the top two rows, real images are depicted, whereas in the bottom two rows, synthetic images are depicted. The latter can be identified by dashed boxes. From now on, artificial images and related plots will be consistently indicated by dashed boxes around them in order to help distinguish them from real measured images. All the images in Fig. 5.1 were selected randomly. As one can see from the figure, synthetic and real images look very much

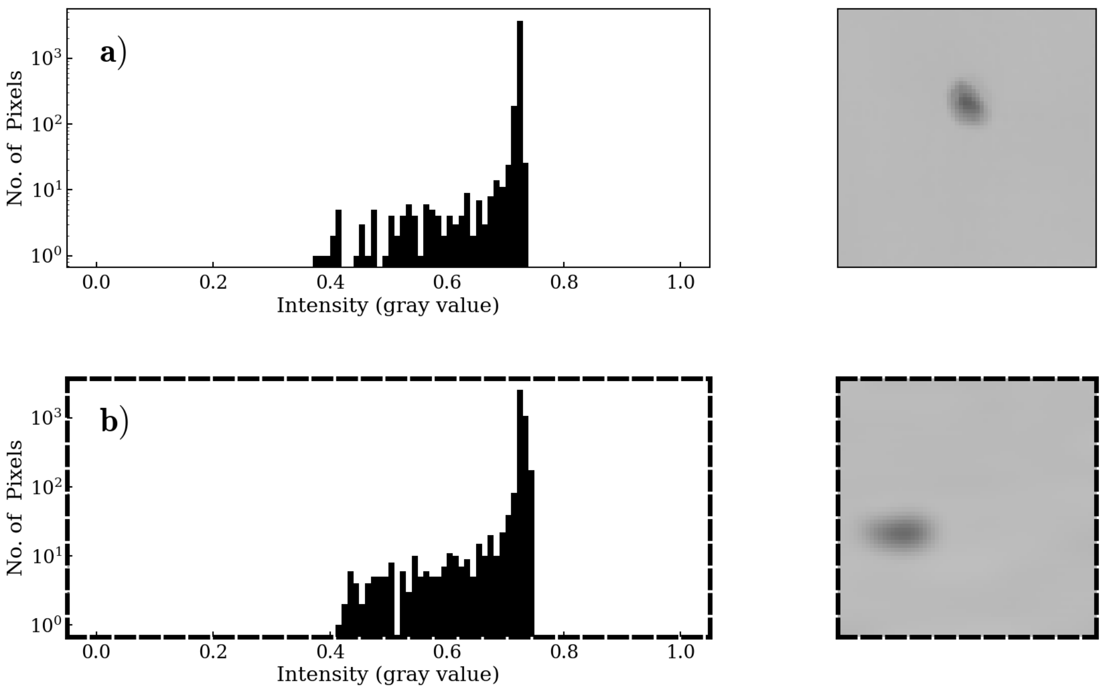


Figure 5.2: *Histogram comparison of a real (top) and an artificial (bottom) blister image. For each case, an image containing one blister is depicted along with its intensity histogram. An intensity value of zero corresponds to a completely black pixel, whereas a value of one corresponds to a totally white pixel. It is apparent that the images not only look similar visually but also correspond nicely with respect to both the intensity range covered as well as the shape of the intensity distributions.*

alike and exhibit only minor differences. In general, they seem to be a bit darker with a less smooth transition in the intensity of the gray value.

However, CNNs may look at the data differently and thus additional checks are required that compare the images in different ways than just visually. One way to do this is to compare their histograms. An example for such a histogram comparison is given in Fig. 5.2. Again, the images look quite alike and, more importantly, also the histograms are fairly similar. Both the intensity range and the shape of the distribution coincide very well. The reader is made aware of the fact that both images in the figure were selected by hand to demonstrate the potential accordance of artificial and real images. However, they are not representative of all the data. It is apparent that the artificial images in Fig. 5.1 exhibit significant variation especially with respect to their brightness. This would become visible in the histograms as a general shift of the distribution to higher or lower values. However, if the variation in the artificial data is larger than in the actual measured data, this is not an issue because the CNN will then just become more robust to this variation in the training data. This may actually increase the performance even on the real data because the network has to learn a more general abstraction in this case. As long as the real data is represented by the artificial data in all major regards, it is reasonable to use it as training data.

In summary, both validation techniques have shown that the artificially generated images resemble the real images very well. Therefore, it is appropriate to use them as training data for the CNNs. However, a final assessment of the quality of the training data can only be made when considering the performance of the networks on the real data (cf. Chapter 9).

Chapter 6

Implementation of the CNNs

This chapter gives an outline of how the CNNs that are used in the present thesis were implemented. This in particular involves the general setup, parameter optimization, and training of the CNNs. In the following, an overview of the general approach to the problem of automated blister detection will be given. Then, the software that was used for the implementation of the networks will be briefly discussed. Afterwards, the setting of the network hyperparameters will be treated along with appropriate optimization schemes. Finally, the results of the optimization procedure for all CNNs will be outlined.

6.1 Approach

The approach that was used for the automated analysis of the image data consists of several distinct steps. At first, small snippets[1] from the large image are selected because the CNNs are better suited for small image sizes. This is due to a number of reasons. On the one hand, the amount of data and the number of parameters can become too large to be processed during training. On the other hand, analyses like the localization of multiple objects of the same type in a single image can cause conceptual problems. When there are multiple objects in the image, the mapping of specific output neurons, i.e. the labels, to the objects is ambiguous. This problem is called label switching and can cause a rapid decline in the prediction quality when the number of objects increases. Hence, the snippets that are passed to the CNNs should contain as few blisters as possible. On the other hand, the snippets need to be large enough to incorporate large blisters well. This is why a size of 64×64 pixels was chosen for the snippets. This is equivalent to an area of 217×638 μm^2 on the sample surface and is twice as large as the largest reference blisters (cf. Section 5.2). Also, choosing a square shape and a power of two for the snippet dimensions makes the CNN faster because it can be parallelized more efficiently on the GPU.

[1]The terminology with regards to different image segments was illustrated in Fig. 4.3 and discussed in the last paragraph of Section 4.1.2.

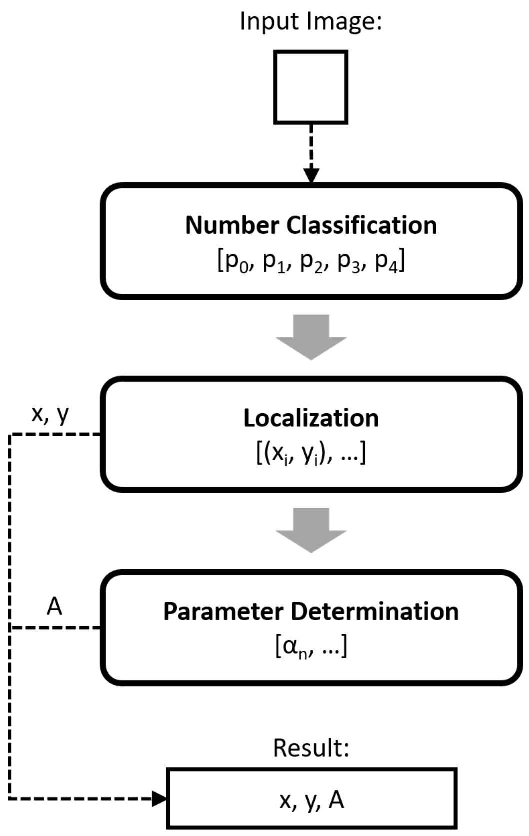


Figure 6.1: *Flow diagram of the automated blister analysis. The input image snippet gets fed to a neural network that classifies the number of blisters in the image. Afterwards, a CNN that is responsible for the respective number of blisters localizes all blisters in the image. Finally, specialised networks identify important parameters of each blister. In the present investigation, the size of each blister as defined in 5.1 is estimated. Hence, the analysis procedure overall yields information on the position and the size of each blister. The identification of further parameters can be added at the final stage and would have to be implemented with additional specialized networks.*

To process the entire slice, the snippets are successively extracted from the image row by row and column by column and the CNNs are applied to each one of them separately. This method is also known as sliding window approach since its application is equivalent to sliding a window of size 64×64 pixels across the image and applying the CNNs at every step. This procedure was already illustrated in Fig. 4.3, when the image segment terminology was defined. The step size between successive snippets is chosen as a fraction of the snippet size to achieve significant overlap of the snippets. This ensures that no blisters are missed that might occur on the border of a snippet. Also, the overlap results in every blister being analyzed several times by the CNNs, which yields a statistic for the blister position and thereby improves its precision. In addition, it enables consistency checks between individual identifications.

The process of automated analysis with CNNs comprises three stages with a total of six individually trained networks. The overall analysis pipeline is illustrated in Fig. 6.1. At the first stage, a CNN estimates the number of blisters that are visible in the image. A maximum number of potential blisters has to be chosen prior to training as it determines the number of neurons in the output layer. It should be realistic so as to not affect the outcome of the training too much. It was adjusted to the highest average blister density in the real data for which detection is likely to succeed without making the detection process too complex. A reasonable value to account for this trade-off is four blisters. The network gives probabilities $[p_0, ..., p_4]$ for all potential numbers as an output, where p_0 is the probability for zero blisters in the image, and $p_1, ..., p_4$ are the probabilities for the cases of one to four blisters, respectively. Depending on the assessment of this first network, the image gets passed on to one out of four CNNs in the second stage which are responsible for the localization of all blisters in the image. There is a separate network for each possible number of blisters from one to four. For the case that there is no blister in the image, no further analysis is needed. The outcome of this second assessment is a list of all blister locations in the image $[(x_0, y_0), ..., (x_n, y_n)]$. At the final stage, each individual blister gets analyzed by a third network which estimates the size of the blister, which is defined as the area it covers on the surface as in Eq. (5.1). For this purpose, a section from the original total image that is centered around the blister's location gets extracted. This simplifies the size estimation, since the blister of interest is always located in the center of the image instead of an arbitrary position. Further networks can be added to this last stage of analysis in order to estimate additional parameters.

This modular approach features some significant advantages. First of all, there are practical reasons to split up the different analysis tasks. Defining an overall loss function for all tasks at the same time is not practical since they represent a mixture of classification and regression tasks. It would be very hard to balance the penalties for all features so as to get equally good results on all outcome quantities. Also, the required network structure might be different for the individual tasks. In addition, if the identification fails, it is hard to identify the reason why that happened. With different stages, it is easier to attribute issues to a certain stage and find appropriate solutions. Adapting the network structure and creating suitable and exhaustive training data are both easier with specialized networks. Moreover, prior knowledge from earlier assessments may be used in the further analysis. For example, the location of the blisters is used to facilitate the estimation of their parameters by choosing an appropriate image section. Finally, also performance reasons make a modular approach more favorable. First of all, the precision of the networks will be better when they can specialize and only have to solve a single task. Moreover, with a modular setup the adaptation to new data or different requirements will be more flexible and easier to implement. Parts of the already existing networks might be applicable to other scenarios as well and do not have to be redeveloped or retrained.

For the entire pipeline, a total of six separate networks were trained. One for the number classification, four for the localization of the blisters and one for the estimation of their size. The training data was adapted to each scenario individually. In particular for the size estimation, images were created in which the blister of interest was placed in the center. The respective position was subjected to Gaussian noise with a standard deviation equivalent to the average precision of the localization, i.e. two pixels. The labels of the training instances for the number classification were one hot encoded which means that the number of blisters is indicated by the position of a single digit '1' in a vector with as many elements as there are potential classes. For performance reasons, the gray values of both the training images as well as the real images were normalized to the [0, 1] interval before they were analyzed by any of the networks.

6.2 Tensorflow: A Machine Learning Framework

For the implementation of the CNNs, TensorFlow 2.0 was used [107]. It is a popular open source machine learning library developed by Google. It is widely used for the development of neural network applications in both research and industry [108]. Since its update to Tensorflow 2.0, it provides a similar functionality as PyTorch, but due to its larger user base and better community support it is preferable to the latter. Version 2.0 features a high-level API called Keras [109]. In combination with the proprietary CUDA framework, Tensorflow can utilize NVIDIA GPUs to parallelize and speed up its computations. For this investigation, an NVIDIA GE-FORCE GTX 1660 Ti GPU was employed. TensorFlow allows developers to create dataflow graphs which represent specific information relations as well as processing sequences and comprise two main components. One are the nodes which describe the individual computational processing steps and the other one are the tensors, i.e. multi-dimensional arrays, which represent the data or information that flows through the graph. Thanks to the Keras API, the user does not have to define these graphs by hand but can straightforwardly set up neural networks at a high conceptual level and implement them in Python. A detailed overview of the concepts and functionalities of both Tensorflow and the Keras API can be found in a book by Géron [37].

6.3 General Considerations on CNN Optimization

When training and optimizing CNNs, there are basically three conceptual levels at which either configurations have to be set or parameters have to be adapted. At the lowest level, there are the network parameters which comprise the weights and the biases of all neurons. The optimization of these network parameters poses a continuous optimization problem because the gradients of the overall network function can be computed with regards to each parameter. Therefore, it can be fully automated with gradient descent algorithms (cf. Section 3.7.4) in Tensorflow and will not be treated this section. In addition to the network parameters, there are also so-called hyperparameters, which have already been referred to in Section 3.7 and who pertain the network architecture. Typical hyperparameters include the number, type, and sequential arrangement of layers as well as the kernel dimensions and step sizes. They ultimately determine the structure of the overall network function and thereby heavily influence the training outcome and the performance of the network. It is hence an important part of the development process to optimize these hyperparameters [110]. Since they pertain the network structure itself, their optimization poses a discontinuous optimization problem. To solve this kind of problem, elaborated search schemes such as Bayesian optimization can be employed. Implementations of these schemes are also available in the Tensorflow framework (cf. Section 6.4). They are used to scan certain hyperparameter ranges and find the best networks therein. However, these automated routines again require the developer to set the values of some optimization parameters (cf. Section 6.6) prior to execution, for example the maximum number of trials the algorithm is permitted to run. These settings then constitute the highest level of configuration in the overall optimization procedure of the CNNs. The hyperparameters of all CNNs in this thesis were optimized with a Bayesian optimization scheme. To employ such an optimization technique, a reasonable initial setup needs to be created. This setup will be discussed in detail in Section 6.5.

The outcomes of the overall optimization procedure of the networks consisted of two distinct types. Some parameters were set to a specific value at the end of the optimization, whereas other parameters were only restricted to a certain range. This was done since the goal of this investigation is to evolve a good hyperparameter range in which successful networks can be found with a high probability instead of just providing a single network that works well on the specific training data. This approach is more suitable for two reasons, one of them being the reproducibility. The outcome of the training procedure is by no means deterministic since the training data was created with randomized settings and the network parameters were initialized randomly. Trying to reproduce a specific network might not yield equally good results. Instead, by providing an appropriate range for the hyperparameters, it is ensured that other researchers can reproduce a well functioning network easily. In addition, working with a parameter range simplifies the development process when the data changes or adaptions are to be made to the network structure for some other reason. It will be easier to find a good network or at least get an indication in which direction to change the hyperparameters in order to improve the network. For both reasons, it is more appropriate to provide a hyperparameter range that can be searched by automated routines that yield concrete and well functioning networks.

As described before, CNNs operate somewhat like a "black box" and it is difficult to predict which setup will yield good results. Hence, it is necessary to try different structural setups to find useful networks. Therefore, the optimization of hyperparameters involves a lot of trial and error. It is an iterative process which starts with an initial set up (cf. Section 6.5). Each iteration involves training many CNN instances with hyperparameters in pre-set ranges and evaluating their performance. The quality is evaluated based on the performance of the networks on a validation data set. Then, other hyperparameters can be added or the accessible value ranges can be adapted accordingly before another iteration is conducted. Thereby, the process iteratively closes in on the best parameter ranges. At the end of the iterative process, the search space for the hyperparameters is already very refined to the range in which the training is successful and yields good results. Basically all parameter ranges where training never led to any useful network are already excluded. In some cases, in which a single value of a hyperparameter turned out to be strictly superior, the range was refined to a fixed value during the optimization process. In the other cases, the range was kept as large as possible to allow for more diverse networks to be found. In yet different cases, structural components or concepts did not yield any useful results and hence were deprecated completely. These included for example residual structures but they will not be regarded further in the present work. As a result, there is not a single correct or best network structure. Instead, within the obtained parameter ranges there is a great variety of networks with very diverse setups which can achieve similar performances and are equally applicable.

So overall, searching for the optimal CNN settings remains a procedure that has to be conducted manually by the developer to some degree albeit it is aided by automated optimization schemes like Bayesian optimization, which help to evaluate different parameter ranges and find specific successful networks efficiently. In summary, the optimization and training of CNNs comprises three distinct conceptual levels. At the highest level of the configuration of the optimization algorithms, the tweaking is done by hand and results in fixed parameter values. At the level of the hyperparameters, the optimization can be largely automated with Bayesian optimization and the results comprise both fixed values as well as parameter ranges. At the lowest level of the network weights and biases, the results are obtained with fully automated routines and consist of concrete parameter values.

6.4 Bayesian Optimization of Hyperparameters

The above described procedure of finding appropriate ranges for the hyperparameters, which poses a discontinuous optimization problem, can be aided by automated optimization schemes like Bayesian optimization. In theory, they can automatically find the best hyperparameter values within a certain pre-defined scope. Primitive approaches like grid search and random search just execute the training of the network for some hyperparameter combinations and rank them by their performance. More advanced optimization techniques like Bayesian optimization consider the evaluation of a number of hyperparameter combinations and adapt the hyperparameters according to an elaborated optimization scheme. The reason why these more advanced optimization techniques are necessary is that there are too many potential combinations and settings of hyperparameters so that not all of them can be tried.

One needs a thoughtful search scheme that takes into account the results of past executions in order to efficiently scan the available parameter space and find suitable settings.

Bayesian optimization is a sequential scheme for the global optimization of black-box functions [111]. It creates a probabilistic model of the relation between the hyperparameters and the objective, which in this case is the performance on the validation data set. Based on this model, it proposes a certain hyperparameter configuration and evaluates it, whereby it tries to gain more information on the aforementioned relation. The results are then used to update the model and start a new iteration. It has been shown that Bayesian optimization yields better results than approaches like grid search and random search and even outperforms human experts at selecting hyperparameters [112]. It comes along as a standard optimization scheme with the Keras Tuner library in Tensorflow [113]. The optimization scheme has to be configured prior to execution. The respective settings will be discussed in Section 6.6.

6.5 Initial Setup

The general setup idea was the same for all of the CNNs in this thesis. It was mainly inspired by examples from text books and successful state-of-the-art CNNs for image recognition, such as LeNet-5 [87] and VGG-16 [114], while also taking into account theoretical considerations. For details, see Section 3.9 on CNNs and the generic review of deep learning from LeCun *et al.* [89]. The initial setup for this investigation widely follows the setup that was already depicted in Fig. 3.6. Apart from the input and output layer, it employs two main structural components which are standard in CNNs. The first one is an alternating series of convolutional and pooling layers at the beginning and the other one is a number of dense layers at the end of the network right before the output layer. More specifically, the first block comprises a combination of convolutional layers with a ReLU activation function and max pooling layers in between, which is a very common architecture. Additionally, in the last block with the dense layers, dropout layers were introduced so as to regularize the network (cf. Section 3.8.3). Another important part of the initial setup is the restriction of the search space of the hyperparameters to reasonable ranges according to conceptual considerations. For example, the number of pooling layers in the CNN is restricted by the image size. The image cannot be downsampled more often than the number of pixels permits.

Apart from these general considerations, a few settings were also chosen according to the specific problem statement at hand. As outlined in Section 6.1, the development of different types of CNNs is advantageous for solving the three distinct tasks in the blister detection pipeline. The pre-chosen settings encompass the number of units in the output layer, the output unit activation functions, and the loss function for each CNN. These parameters are determined by the inherent structures of the tasks and will not be altered during the optimization process. For details, the reader is pointed to the appropriate sections in Chapter 3.9. For the first network, which is the number classification, five output units were used. Each unit represents one of the five possible cases of zero to four blisters in the image. A softmax activation function was applied which is typically used for pure classification tasks [115]. Thereby, the outputs can be interpreted as classification probabilities for the different cases. The appropriate loss function is the categorical cross entropy. For the localization stage, in total four separate networks were trained, one for each case of one to four blisters. For each blister in the image, two output neurons are used in the output layer to represent its x and y positions. The activation function is sigmoidal since the dimensions for both the x and y directions were scaled and restricted to a (0, 1) interval, which is equivalent to the width and the height of the image, respectively. However, the loss function for this problem had to be customized for a couple of reasons. The general notion of these issues will be outlined here but they will not be discussed in detail. One of the problems is that the network does not know which blister in the image is supposed to correspond with which output neurons. This problem is called label switching. The solution is to consider all permutations of the blister instances and compute the loss for all of them. Afterwards, the smallest loss can be regarded as the loss of the prediction of the network. Another problem to consider is that the choice of penalization for the deviations from the predicted values is not unambiguous. Depending on the overall identification performance, one might for example be interested in minimizing the largest deviation of a single blister or one could focus more on the average deviation for all blisters. Hence, the choice heavily depends on what predictions and accuracies the task requires and how the developer wants to penalize certain outcomes. To be able to dynamically adapt this emphasis made by the loss function, it must be explicitly coded by the developer. The implementation of the loss function in Tensorflow is rather involved and will not be considered further at this point, however the details can be gathered from the code listings in Appendix C. Finally, for the estimation of the size of the blister, a single output neuron suffices. No activation function was applied to the output neuron so as to not restrict the potential sizes of the blisters since the purpose of the network is just the regression of a single value. For such standard regression problems, the choice of mean squared error as the loss function is appropriate.

6.6 Optimization Results

After the initial setup was created as outlined above, the iterative optimization procedure described in Section 6.3 was applied. The respective implementation in Python can be found in Appendix C. The results of the optimization will be presented top down meaning that more general parameters will be presented first and more specific parameters regarding the network structure will be presented later. As explained before, the results will in general consist of parameter ranges, except for cases where single values were performing strictly better. When optimizing the hyperparameters, just testing random parameter combinations and settings will not be efficient. One has to make reasonable propositions as to what specifications might yield good results. A lot of different concepts and hyperparameter settings were explored. Ideas were for example taken from successful state-of-the-art CNNs just as was done for the initial setup in Section 6.5. Additionally, the parameters were incrementally and intuitively improved after considering the performances of the networks from the last iteration on the validation data set. It should be noted that several hyperparameters and concepts, such as residual structures, were tried but will not be employed in any of the final networks. Those trials will not be considered further in this section.

Parameter	**Value**
Maximum number of trials	64
Number of initial points	16
Executions per trial	1
# training instances	16384
Validation split	0.125
Number of epochs	128
Batch size	64

Table 6.1: *General optimization parameters concerning the optimization of the hyperparameters as well as the network training. The first three parameters are settings that influence the behaviour of the Bayesian optimization scheme. The others are global parameters that influence the general training process.*

The same approach and optimization procedure were used for all three stages of the blister identification pipeline, which was described in Section 6.1. The optimal values for both the fixed parameters as well as the parameter ranges turned out to be fairly similar for all six CNNs. For the sake of simplicity, the final values and ranges were set to be identical for all of them. Advantages of leaving a range of the parameters instead of fixing their values where possible were already outlined in Section 6.3. The results of the hyperparameter optimization are summarized in separate tables. For example, Tab. 6.1 displays, among others, the results of the tweaking of the Bayesian optimization search parameters. The maximum number of trials defines the total number of trials which are conducted by the optimization scheme. The number of initial points describes how many points in parameter space the Bayesian optimization algorithm considers before it starts to adapt its model and infer which parameter configurations to try next. For each trial, just one execution of the training was conducted. The other parameters in the table concern the general Keras Tuner optimization parameters. A total of 16,384 training instances were generated with POV-Ray for each scenario (cf. Section 5). The validation split of 0.125 means that 1/8 of the original data set was held out as a validation set for evaluating the networks performance. Hence, 14,336 instances were actually used for training. A total of 128 epochs were conducted meaning that the entire training data set was used 128 times for training. During each epoch, a batch of 64 training instances was considered at a time before updating the weights.

Hyperparameter	**Pertaining**	**Value**
Padding	all conv	same
Stride	all conv	(1, 1)
Kernel size	conv1	7x7
Kernel size	conv2 - conv_max	3x3
Activation	all except output	ReLU
Max Pooing kernel	all max_pooling	2x2
Max Pooing stride	all max_pooling	(2, 2)
Regularization	all	0

Table 6.2: *List of hyperparameters for which a range of values was explored but only a single value remained after refinement. The network components to which the hyperparameter pertains as well as the respective value are given.*

Tab. 6.2 gives an overview of the hyperparameters for which different values and ranges were investigated but only a single best value was chosen after the optimization. First of all, the padding of all convolutional layers was selected to be ’same’. With this setting, the input images are padded with zeros around them in order to retain the original shape of the image or feature map. Also, the best stride for the convolutional layers turned out to be 1×1. Thereby, the kernel is shifted just one pixel at a time across the image in both x and y directions and is applied at every location. This means that the resulting feature maps have the same dimensions as the input. As is often the case in state-of-the-art CNNs, the kernel size of the first convolutional layer is larger then for the rest of the convolutional layers. For the first layer it is 7×7, whereas in all successive convolutional layers it is 3×3. The best activation function for all layers including the convolutional and dense layers but excluding the output layer was the rectifying linear unit, which was discussed in Section 3.5. The kernel of all maxpooling layers had a size of 2×2 pixels with a non-overlapping stride of 2 in both x and y directions. With this choice, both dimensions of the feature maps get halved during each maxpooling layer. Finally, the employment of a regularization term as discussed in Section 3.8.3 in all layers

Hyperparameter	**Pertaining**	**Minimum**	**Maximum**	**Step**
Stride	conv1	1	2	1
# Filters	conv1	32	96	32
# Add. conv layers	-	8	16	1
# Add. MaxPool layers	-	0	4	1
# Filters	conv2 - conv6	32	192	32
# Filters	conv7 - convMax	32	192	32
# Dense Layers	-	1	3	1
# Units	dense1 - dense3	64	384	64
Dropout Rate	-	0.1	0.5	0.1
Learning Rate	-	0.00002	0.0005	varying

Table 6.3: *List of hyperparameters for which the optimization resulted in a range of values. The network components to which the hyperparameter pertains are given. Also, the minimum and maximum values of the hyperparameter are stated along with a step size which determines the resolution of the search space that the optimization scheme can access.*

did not improve the performance of the CNNs on the validation data set although a large variety of magnitudes was tried.

Tab. 6.3 gives a collocation of the hyperparameters for which the optimization resulted in a parameter range rather than a single superior value. Just as for the kernel size, the values for the stride and the number of filters for the first convolutional layer deviates from the rest of the convolutional layers. The stride can be either one or two for both the x and y direction, while the number of filters can range between 32 and 96. The number of additional convolution layers apart from the first one may range between 8 and 16. The optimal number of additional maxpooling layers can range between zero and four. The additional convolutional layers were split into two sections with separate parameters for the respective number of filters to allow for more flexibility in the networks. However, the range of optimal values turned out to be the same for both of them, namely 32 - 192. The number of dense layers at the end of the network structure was varied between one and three with similar performance results. The number of units in each of these layers can be set between 64 and 384 without significantly influencing the prediction outcomes. As another means of regularization, a dropout layer was added after the last dense layer. Its dropout rate can be chosen from the interval of 0.1 to 0.5. For the learning rate of the training algorithm, the optimal values were refined to a range of $2 \cdot 10^{-5}$ to $5 \cdot 10^{-4}$ with varying steps in between.

In order to acquire specific networks as required, a Bayesian optimization search is executed for each of the six necessary CNNs. The optimization parameters listed in

Tab. 6.1 and the hyperparameter values and ranges discussed above are employed. The search scheme results in a variety of networks sorted by their performance on the validation data set. The entire optimization procedure for one type of network takes about 20-30 h on a CUDA-enabled NVIDIA Geforce GTX 1660 Ti GPU. However, the training of a single specific network can be done in less than half an hour. The procedure usually yields a number of good networks with similar performance but very different setups and parameters. For the automated blister analysis later on, the respective best networks are used. However, this does not imply that their structure is necessarily superior to the setups of the other networks. The differences in performance can also be due to different random initializations for example. The parameter ranges determined before define a reasonable search space and any network within these ranges has a viable structure and is likely to lead to a good performance.

Chapter 7

Analysis of CNN Precision on Artificial Data

After setting up and training the neural networks, a first investigation of the performance was conducted on artificial data. To this end, a separate test set of artificial data consisting of 616 instances was created for each of the networks. These were then used to quantify the precision of each CNN. In the following sections, the precision will be separately investigated by different means for each of the three stages of the blister identification process. The results are promising, in particular for smaller blister densities, and make it seem likely that all of the networks will be applicable to real data as well.

7.1 Number Classification

For the precision analysis of the number classification, the true number of blisters was compared to the prediction made by the CNN for each test image. In order to give a comprehensive overview of the results, a classification error matrix was computed. In this matrix, which is given in Tab. 7.1, the aggregate number of cases for each combination of the true and predicted number of blisters is given. Originally, the output of the network comprises probabilities for all five cases. Of these probabilities, the maximum was selected as the prediction of the network. In fact, this reduction of information was unambiguous since the maximum value was above 50% in all cases. As is discernible from the table, the number classification works very well for up to two blisters and only occasional misclassifications occur. For three or more blisters, the number of falsely classified images increases to some degree. However, by applying the CNNs to the real data via a sliding window scheme, which has already been introduced in Section 6.1, the accuracy of both the number classification and the localization of the blisters can be improved significantly. This will become apparent in the analysis of the real data in the next chapter.

		Predicted Number				
		0	**1**	**2**	**3**	**4**
True Number	**0**	121	0	0	0	0
	1	0	125	0	0	0
	2	0	1	112	3	0
	3	0	0	3	123	14
	4	0	0	1	7	106

Table 7.1: *Classification error matrix of the blister number estimation. The rows indicate the true number of blisters in the image, while the columns represent the prediction made by the CNN. Entries on the diagonal hence represent correct classifications. The table shows that the accuracy of the number classification is extraordinarily high for up to two blisters per image snippet but begins to decrease for three or more blisters. However, the accuracy for high blister densities is later improved by the application of the CNNs via a sliding window scheme.*

Overall, the predictions made by the number classification CNN can be considered very reliable, in particular for low blister densities. However, if thorough investigations and precise analysis are to be conducted for high blister densities, the CNN may have to be improved.

7.2 Localization

With regards to the precision of the localization, two quantities are of interest. On the one hand, the average deviation of the blister locations from their true location should be as small as possible. On the other hand, the localization of some blisters should not be improved at the cost of deteriorating the precision of other blisters. In particular, the largest deviation in an image should not get too large even if the other blisters are localized very well in return. Hence, the optimization must be balanced for both of these goals. To this end, the average mean error and the average maximum error in pixels were computed across all of the test instances to analyse the precision of the estimations made by the CNN. Tab. 7.2 lists the respective values itemized by the number of blisters. As is apparent from the table, the localization for the one-blister case reaches sub-pixel precision. Also for the two- and three-blister cases, the average error is around one single pixel but it increases to two pixels in the four-blisters case. The average maximum error increases faster and reaches up to four pixels. However, compared to the optical resolution of the setup of 2.76 pixels, these values are still relatively small. The decrease in precision for increasing blister densities is largely due to cases in which blisters lie very close to each other or

# Blisters in Image:	**1**	**2**	**3**	**4**
Average Mean Error	0.44	0.82	1.25	2.03
Average Maximum Error	0.44	1.09	2.20	4.08

Table 7.2: *Precision analysis of the localization networks. The averages are computed across all considered test instances. The mean error is the error averaged over all blisters in the image snippet, while the maximum error is the error of the blister in the snippet that has the highest deviation from its true position. The errors are given in units of pixel. Compared to the optical resolution of 2.76 pixels, all values are relatively small, in particular those for up to three blisters. The accuracy of the localization is improved even further by the application of the CNNs via a sliding window scheme, as described in Section 6.1.*

overlap. This can be examined in Fig. 7.1 and Fig. 7.2 in which sample images with three and four blisters are depicted along with the true and estimated positions of all blisters. With three blisters in the image, the localization works very well even in the aforementioned difficult cases. With four blisters, however, the mean deviation increases and difficult cases do not get identified very well. However, similar to the number classification, the accuracy of the localization for high blister densities can be improved significantly by the sliding window approach that is taken for the application to the real data. Nonetheless, for higher blister densities with more than four blisters per image snippet, the performance would decrease significantly and a different approach would probably be required.

7.3 Size Estimation

Finally, the precision of the blister size estimation will be examined. For the training of the respective network, special training data was created. The blister of interest was placed in the middle of the image and some noise was applied to the position. The noise was implemented with a Gaussian distribution, the standard deviation of which was equivalent to the highest average mean error of the localization. As described in the previous section, this value was two pixels in the case of four blisters (cf. Tab. 7.2). Afterwards, a random number of blisters was placed in the scene so as to cover all possible cases. The size of the blister, which was defined as the covered area on the surface in Eq. (5.1), is estimated as if viewed from the top to make it independent of the angle of view of the camera, which in the present case was about 20 degrees. Compared to the original image, as for example in Fig. 4.2, this introduces a constant factor of 2.94 in the size. This is done because by applying

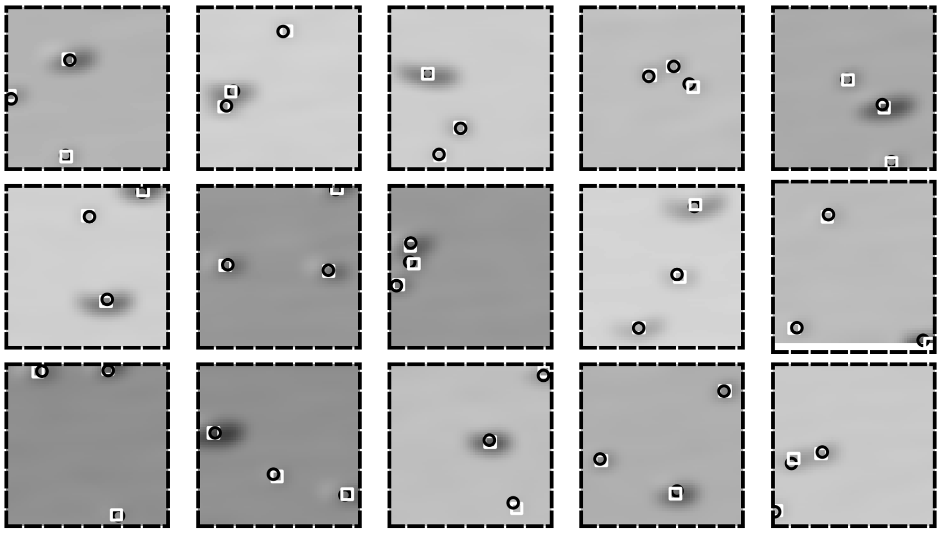


Figure 7.1: *Set of some artificial images with three blisters. The white rectangles give the true position of the blisters and the black circles indicate the prediction of the CNN. The localization works very well even in difficult cases where blisters overlap or blisters lie very close to the edge of the image.*

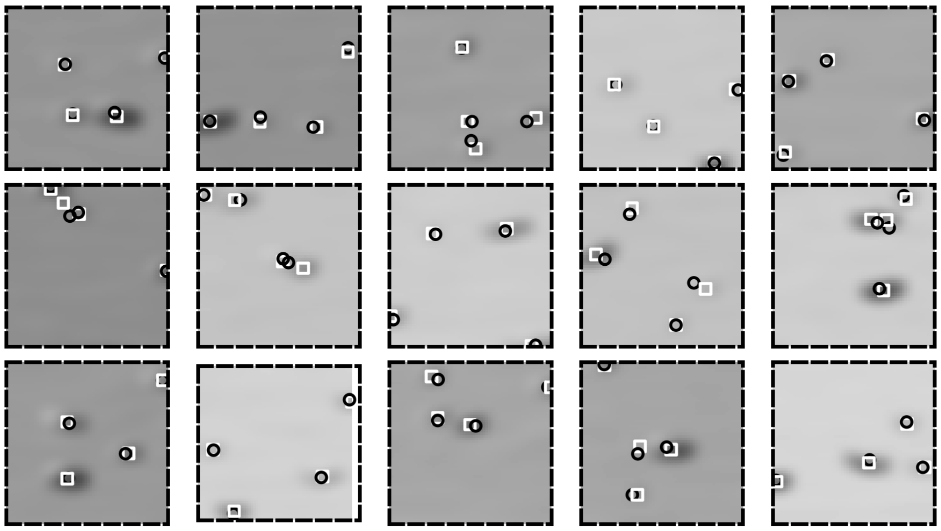


Figure 7.2: *Set of some artificial images with four blisters. The white rectangles give the true position of the blisters and the black circles indicate the prediction of the CNN. The localization works quite well in cases where the blisters are clearly distinguishable but struggles when blisters lie close to one another or overlap. The overall precision is significantly lower than in the three-blister case.*

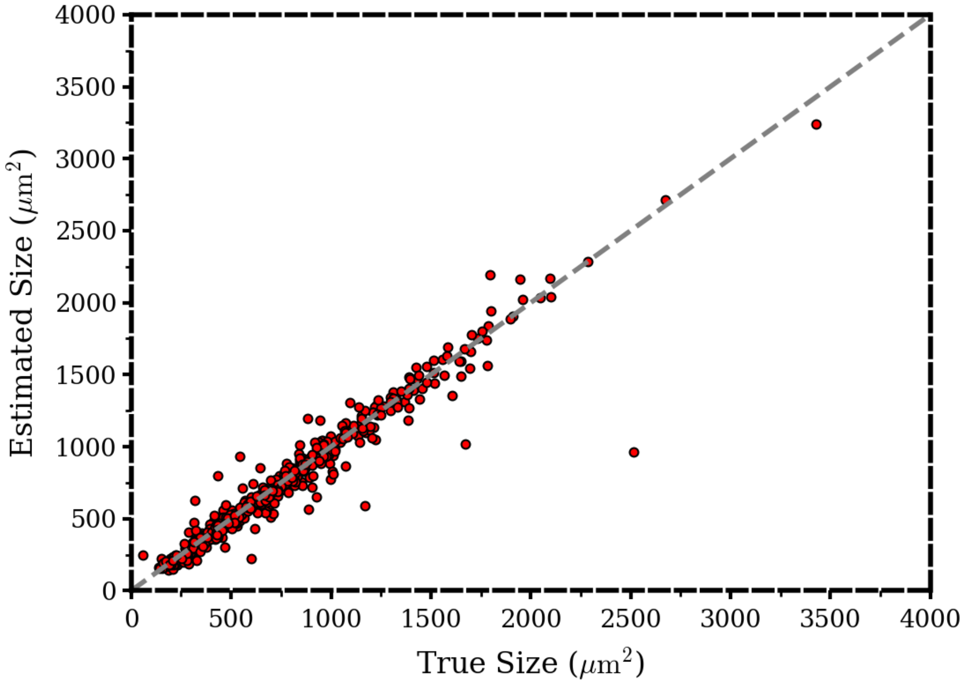


Figure 7.3: *Precision analysis of the size estimation of individual blisters. The true size, which was defined as the area the blister covers on the surface in Eq. (5.1), and the estimated size of the blisters coincide very well across the entire scale. Some outliers occur due to overlapping blisters which cannot be distinguished by the network. A total of 616 test instances were considered for this assessment.*

this implicit adaptation of blister size in the training data the CNNs can learn to directly deduce the true size of the blisters from the side view.

In order to visualize the precision of the size estimation CNN, the estimated size of each blister was plotted against its true size. The result of this analysis is depicted in Fig. 7.3. Due to the telecentricity of the camera lens, a pixel always corresponds the same area in this top-view configuration regardless of the distance. A single pixel is equivalent to an area of 10.69 μm^2 in both dimensions. In the figure, the diagonal where true and estimated sizes coincide is indicated by a gray dashed line. It is apparent that most of the test instances lie on or close to this diagonal. Moreover, the estimation of the size works equally well across the entire scale, at least in the relevant range covered by the test instances. There are some exceptional cases where the prediction deviates significantly, which is mostly due to overlapping blisters. A quantification of the absolute relative deviation of the estimated size from the true size yielded an average value of 8% across all test instances, including the outliers. The number of blisters in the image does not seem to impair the identification process significantly unless a blister interferes with the central blister. Overall, it can be said that the CNN responsible for the estimation of the blister size seems to work very reliably.

Conclusion

Overall, the precision of the number classification and localization CNNs on the test images is very high, in particular for low blister densities up to two blisters per 64×64 pixel image snippet, and decreases somewhat for higher densities. The size estimation yields good results regardless of both the size of the blisters and the blister density in the image. The identification pipeline is hence performing well on artificial data and is expected to do so on real data as well, since the training images are fairly similar to the real images (cf. Section 5.3). However, a final assessment of the performance can only be made when considering real data. This will be provided in Section 8.1.2.

Chapter 8

Analysis of Real Data

In this chapter, results of the automated blister detection in real image data will be discussed. Section 8.1 will outline how the CNN estimations are used to identify blisters in individual image slices, estimate their size, and keep track of them across time and will also include an assessment of the identification accuracy. Section 8.2 will present results that were acquired during the analysis of isolated image slices which did not require the tracking of individual blisters across time (static analysis). Section 8.3 will then discuss results of the analysis that involved the tracking of individual blisters over time and therefore allows to investigate the dynamics of their formation (time-dependent analysis). The results presented in this chapter are meant to illustrate the capabilities of the method and what kind of investigations it enables. Nonetheless, the results acquired so far already yield new insights and indicate that the developed method has the potential to significantly contribute to a better understanding of blister formation.

8.1 Identification Procedure and Accuracy

This section will first describe the procedure along which the CNNs are applied to the real image data and how their detections and estimations are used to identify and characterize individual blisters. Subsequently, the results of a manual assessment of the identification accuracy will be discussed. All source codes related to the identification of blisters are provided in Appendix E and the images used for the accuracy assessment are given in Appendix F.

8.1.1 Approach

The approach that is taken to apply the CNNs to the real image data has been described in detail in Section 6.1. The main procedure involves the analysis of individual image slices according to a sliding window scheme. The detections made by the CNNs during this procedure are evaluated collectively to achieve accurate blister

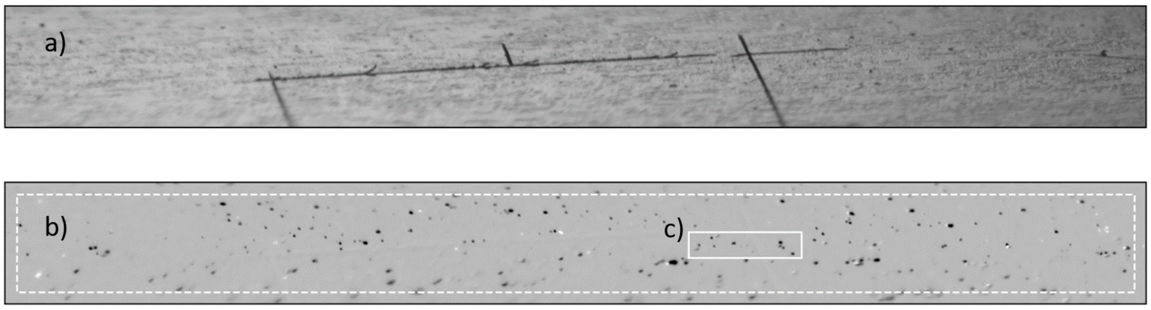


Figure 8.1: *Panel a) depicts the original image slice and in panel b) the background subtracted image is given that is used for the automated analysis. The specimen was exposed to a fluence of* $6.6 \times 10^{23}\ \mathrm{Dm}^{-2}$ *at the time of acquisition of the image. The dashed box in b) indicates the region in which blisters are identified. Outside of this region, the identification procedure disregards all detections made by the CNNs. The image section indicated by c) is used in Fig. 8.2 to illustrate the identification process in more detail.*

identifications. The step size of this sliding window procedure was selected to be eight pixels in both x- and y-direction which means that each pixel is analyzed 64 times. Thereby, multiple redundancy in the detection of blisters is achieved and the precision of the identification is increased. The process is illustrated in Fig. 8.1 and Fig. 8.2.

Each of the 64×64 image snippets[1] considered in the sliding window procedure is subjected to the same analysis procedure. First of all, the number classification and localization networks are applied. The estimated positions of all blister detections are aggregated to a heat map, which is depicted in Fig. 8.2b. For each new detection, the corresponding pixel value of the heat map is incremented by one. This map represents the basis for the actual identification of individual blisters. In the heat map, all local maxima are identified and then analyzed individually. For that purpose, a circular area with a radius of seven pixels is selected around the local maximum. The mean position and standard deviation of all blister localizations in this area is computed along with their total number. Afterwards, the size of the blister, defined as the area it covers on the surface (cf. Section 5.2), is estimated using an additional snippet from the original background-subtracted image (Fig. 8.2a) that is centered around the estimated position. This snippet is passed to the size estimation CNN that is trained to only regard the blister that is located in the center of the snippet. Due to the use of this additional snippet, positions close to the borders of the image slice have to be disregarded. Hence, the region where blisters can effectively be identified

[1]The terminology of image segments used in the present thesis was defined in Fig. 4.3.

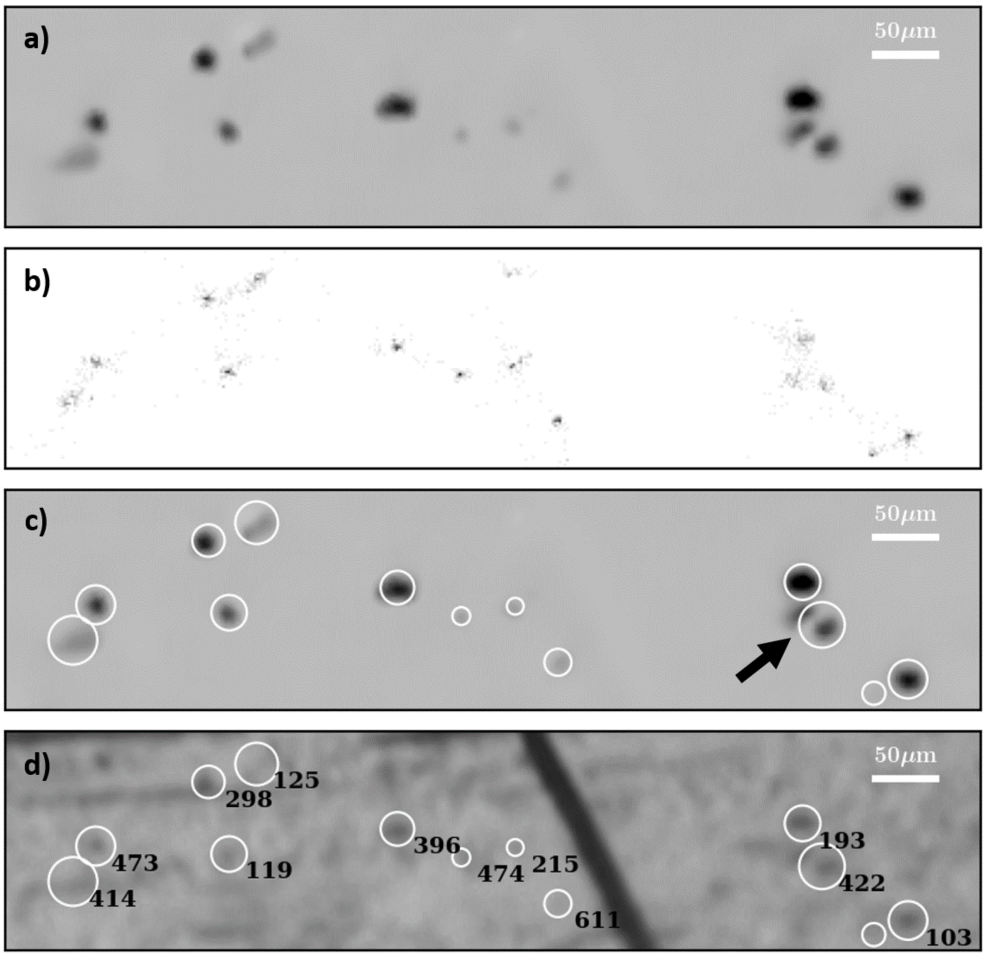


Figure 8.2: *Illustration of the blister identification process based on the image section indicated in Fig. 8.1c. Panel a) is the background subtracted image (90×400 pixels) that is used as input for the CNNs. Panel b) is the heat map of blister detections in which the gray value indicates the cumulative number of detections at this location that are made by the CNNs during the sliding window application. A darker value represents more detections. Panel c) is the same as panel a) but superimposed with the blister identifications, which are indicated by white circles. The center of the circle is located at the estimated position and the area of the circle is chosen such that it is identical to the estimated area that the blister covers on the surface, as given by Eq. (5.1). The arrow in panel c) indicates a case where the procedure was not able to separate two overlapping blisters. Panel d) displays the same image section as panel c) but in the original image without the background subtracted. It illustrates how sensitive the overall identification framework is. The annotated numbers for each blister are the record identifiers.*

is somewhat smaller than the original image slice. This region is equivalent to an area of 15.6 mm^2 and is indicated by the dashed box in Fig. 8.1b. The omission of blisters outside of this region does not constitute an issue since there is significant overlap between consecutive image slices. Hence, blisters that lie in the border region of one slice will be in the central region in the next slice and hence will be considered at least once. In order to increase the precision of the identification, the processing of the snippets described so far is repeated in a second iteration that builds on the results of the first iteration. The aggregation region in the heat map is now centered around the previously estimated position and its size is equivalent to the estimated size of the blister. The second iteration involves an update of the size estimation as well. Finally, the sum total of the detections η, which are aggregated in the second iteration, is used as a decision criterion with an empirical threshold value of $\eta_{th} = 25$ to make a final assessment on the identification of the blister according to

$$\text{if } (\eta > \eta_{th}) \text{ then : add Blister} \qquad . \tag{8.1}$$

At the end of this procedure, both the estimated location and the size of the blister are known. Those localizations in the heat map that were considered in the second iteration are disregarded in the analysis of further blisters thereafter in order to not let them contribute to multiple blisters. The procedure is applied successively to all the local maxima in the heat map. The final result of this entire process is illustrated for a small image section in Fig. 8.2c. Overall, the location and the size of the blisters seem to be identified very well. An additional, quantitative analysis of the overall identification accuracy is discussed in the next section.

In order to investigate the dynamic properties of individual blisters, they also have to be tracked consistently over time. This means that blister identifications in one cycle need to be matched with the appropriate identifications of the same blisters in the previous cycle. To achieve this, the identification results of all image cycles are analyzed in order. For all blisters in a cycle, it is checked whether an old blister in the previous cycle existed at a position that lies inside the area of the currently considered blister. If so, the identification results of the blister in the current cycle will be assigned to the old blister and added to its record. Otherwise, a new blister instance containing the respective data is created in the data base. The overall result is a complete time-line of the position and the size of each blister. This tracking of individual blisters allows to conduct a dynamic analysis of their formation and evolution across their entire lifetime. The results of this time-dependent analysis will be discussed in Section 8.3. It is to be noted that the entire identification procedure takes up to 30 h on a standard CPU for the analysis of all image cycles in one slice. The majority of this time is consumed during the computations of the CNNs and is mainly caused by the small step size of the sliding window procedure. However, the algorithms were not yet optimized for speed and the detection can be fully parallelized so that significant performance gains can be achieved by these means.

Analysis of Identification Accuracy

Case	Absolute Count	Relative Count
Total Number of Considered Instances	221	100%
Correct Classifications	177	80%
False Positives	5	2%
False Negatives	10	5%
Two Classifications on One Blister	27	12%
One Classification on Two Blisters	2	1%

Table 8.1: *Results of the accuracy assessment of the identification framework. It was conducted for a fluence of* 9.6×10^{23} Dm^{-2}*, for which the training data was optimized. The image sections that were used for this assessment are provided in Appendix F. The results show that the precision is quite good at this fluence, apart from a significant number of cases where one large blister was identified as two separate blisters. The accuracy of the identifications is better at smaller fluences when the blister density is smaller, but increases for larger fluences where the framework was not adapted to the specific blistering parameters.*

8.1.2 Assessment of Identification Accuracy

After implementing the analysis procedure described above, the accuracy of the identifications made by the automated framework needs to be assessed. A first analysis of the precision of the CNNs on artificial data was conducted in Chapter 7. This verification is now complemented by a validation of the results on real image data. For this purpose, an image slice was evaluated manually to compare the identifications by the automated framework with the real blister features. Specifically, several image sections at a fluence of 9.6×10^{23} Dm^{-2} were analyzed, which already served as reference for the parameterization of the CNN training data. These image sections are provided in Appendix F. A total of 221 blister instances were considered and the results are given in Tab. 8.1. All instances were assigned to one out of five categories to evaluate the accuracy. The large majority of 80% of the instances are identified correctly. The fraction of false positives is very low, however, it can be up to 10 times larger in some isolated image cycles when the alignment of the images did not work as well as for the others. Then, some false positives are identified on the position of the marker that becomes visible. This does not pose a serious problem, since the marker position is known and respective cases could in principle be excluded from further analysis. However, with the particular specimen at hand, this was not feasible since the marker runs through the center of the image slice and a significant portion of it would have to be disregarded. However, this limitation can be alleviated in future experiments by placing the marker closer to the borders of the frame. Out of the small number of false negatives, most of the instances were small blisters that lay close to a larger blister and were therefore not recognized by the identification framework. The largest error rate of 12% occurs when two identifications are placed on a single blister. The large majority of these cases are due to large blisters with an irregular and longish shape. A significant amount of these misclassifications could be avoided by adding according shapes to the training data for the CNNs. Finally, the number of cases in which two distinguishable blisters were identified as a single blister is vanishingly small. However, some of these instances coincide with false negatives since they are not always clearly separable from one another. This quantitative analysis of the identification accuracy was complemented by a more qualitative assessment by the author based on the images provided in Appendix F and Appendix D. Together, these analyses have shown that the identification and localization works perfectly for small and isolated blisters and performs well for large blisters as well unless they exhibit a very irregular shape or overlap. The size estimation of the blisters works well for blisters with an elliptical shape, including all small blisters which typically have circular shapes and are predominant for low fluences. However, the precision decreases when blisters overlap or are ill-shaped.

For all three networks, the main causes for misclassifications or lower precision of the estimates are the same. First of all, there are inherent boundaries to the blister parameters that are constituted by the parameterization of the training data, which was adapted to a fluence of 9.6×10^{23} Dm^{-2}, as described in Section 5.2. For blisters not adhering to these boundaries, a reduced performance is to be expected from the CNNs. Major restrictions are the maximum density of blisters, the maximum and minimum blister sizes, and the blister shapes that were included in the training data. Unfortunately, there is no straightforward way to determine these quantities. This is because the identification accuracy heavily depends on the specific interplay of the density, overlap, shapes, and sizes of blisters. In addition, the resolution of the images sets a principle lower boundary for the detection and characterization as well. With an optical resolution of $r \approx 5$ μm and a corresponding blister radius of $R_{min} = 2.5$ μm, the minimum size of a blister A_{min}, that can theoretically be detected, can be estimated as

$$A_{min} = \pi R_{min}^2 \approx 20 \ \mu\text{m}^2 \quad . \tag{8.2}$$

The second main constraint on the accuracy is posed by the inherently difficult cases of overlapping blisters, blisters that have merged over time, or blister that have very irregular shapes. The probability for all of these cases to occur increases with fluence since blister features tend to become larger and is amplified by the increasing blister density. Finally, the visibility of the marker on the specimen in image cycles that were not aligned equally well frequently causes some misclassifications as well. However, as discussed above, this problem can be solved by relatively simple means.

Similarly, the inadequacy of the parameterization of the training data for the analysis of high fluence images can be resolved by expanding the respective parameter ranges. Nonetheless, the problem of idiosyncratic blister features remains and may require some substantial adaptation of the identification framework.

Overall, the accuracy assessment conducted here indicates that the estimations of the automated blister identification framework developed in the present thesis are very reliable and can be used for the intended large-scale statistical analysis of blisters in Section 8.2 and Section 8.3. However, this assessment is only valid for fluence values up to 9.6×10^{23} Dm^{-2}. Thereafter, the accuracy is likely to decrease to some degree since the parameters of blisters start to exceed the ranges that the CNNs were adapted to. In order to get an intuition for how well the identification works at different fluences and corresponding blister densities and what kind of blister features cause specific kinds of misclassifications, it is instructive to consider some illustrations of the identification results. To this end, a number of sample images is provided in Appendix D.

8.2 Results of Static Analyses

This section presents the results of the static analysis, which did not require the tracking of individual blisters across time. The blister density and total surface coverage will be investigated as well as the distribution of blister sizes. In addition, an analysis of the nearest neighbor distances of blisters is conducted to assess whether blisters have an observable impact on blisters in their vicinity.

8.2.1 Blister Density and Total Surface Coverage

A fundamental quantity that can be used to describe the blistering process on metal surfaces is the total number of blisters per area. With the time resolved information on this quantity, the dynamics of blister formation can be investigated. To this end, the evolution of the number of blisters per area depending on the fluence was calculated (cf. Fig. 8.3). The number of blisters per area grows with an increasing rate at the beginning and transitions into linear growth for medium fluence values. For large fluences above 10×10^{23} Dm^{-2}, the total number of blisters per area begins to saturate. Notably, an analogous trend was already found for blisters on tungsten samples by Manhard [22]. Even the fluence value at which the saturation begins matches astonishingly well. However, the analysis on tungsten were conducted manually for only six different fluence values and separate specimens. The comparability of the two studies is therefore limited.

The black vertical line in Fig. 8.3 indicates the fluence that was considered as reference for the optimization of the identification framework and the creation of the

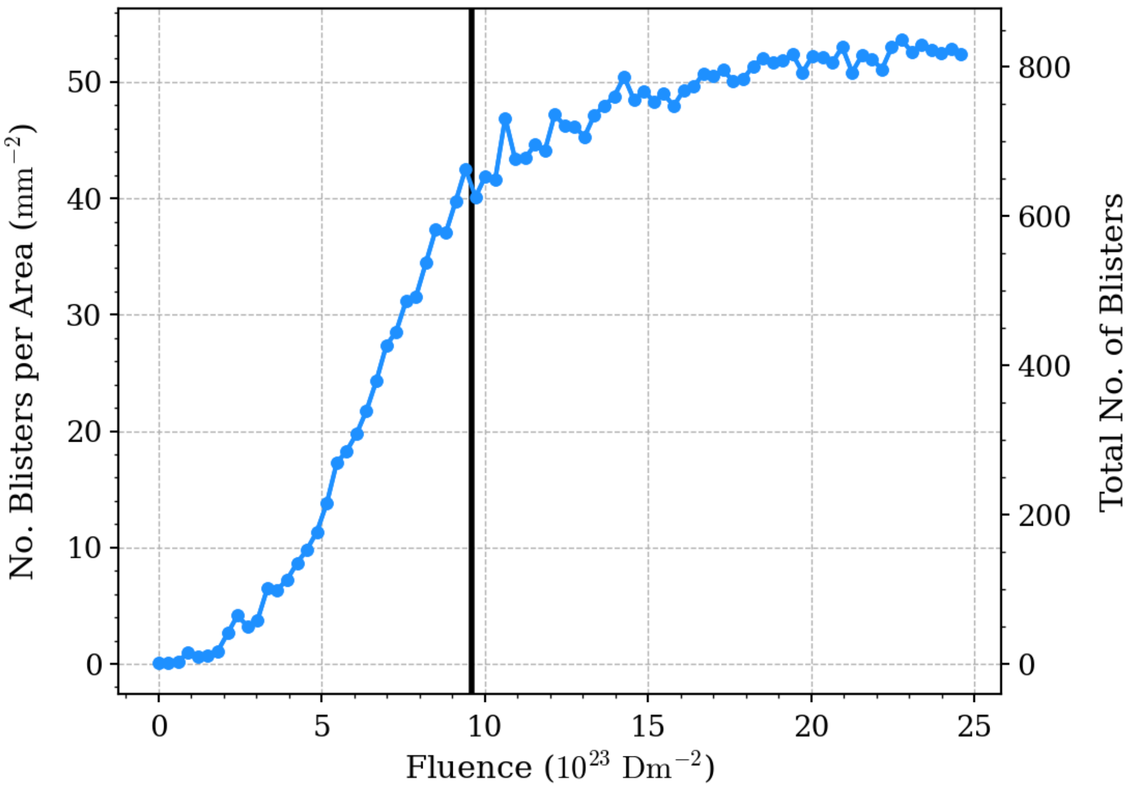


Figure 8.3: *Number of blisters per area as function of fluence. The black vertical line indicates the fluence that was used as reference for the adaptation of the identification framework. For larger fluences, the estimates are becoming less precise due to an increasing number of idiosyncratic blister features, but are still representative of the overall trend. The total area was* 15.6 mm^2.

training data, as described in Section 5. In particular, the CNNs were optimized for the blister parameter ranges at this fluence. As discussed in Section 8.1.2, the quality of the identification deteriorates for higher fluences because more and more blisters evolve beyond the range of the training data set. Consequently, the accuracy of both the number of identified blisters and the estimated size of the blisters decreases. However, despite the lack of appropriate training data, the CNNs and the entire detection pipeline still perform unexpectedly well in the range beyond the black line where both the blister density and the blister size exceed the respective ranges that were considered during training. In Appendix D, some sample images of the identification results for high blister densities are given to provide illustrative support for this claim. Quality measures that will be discussed in detail at the beginning of Section 8.3 support this assessment. The acceptable reliability of the blister identifications at high fluences is partly achieved by the application of the

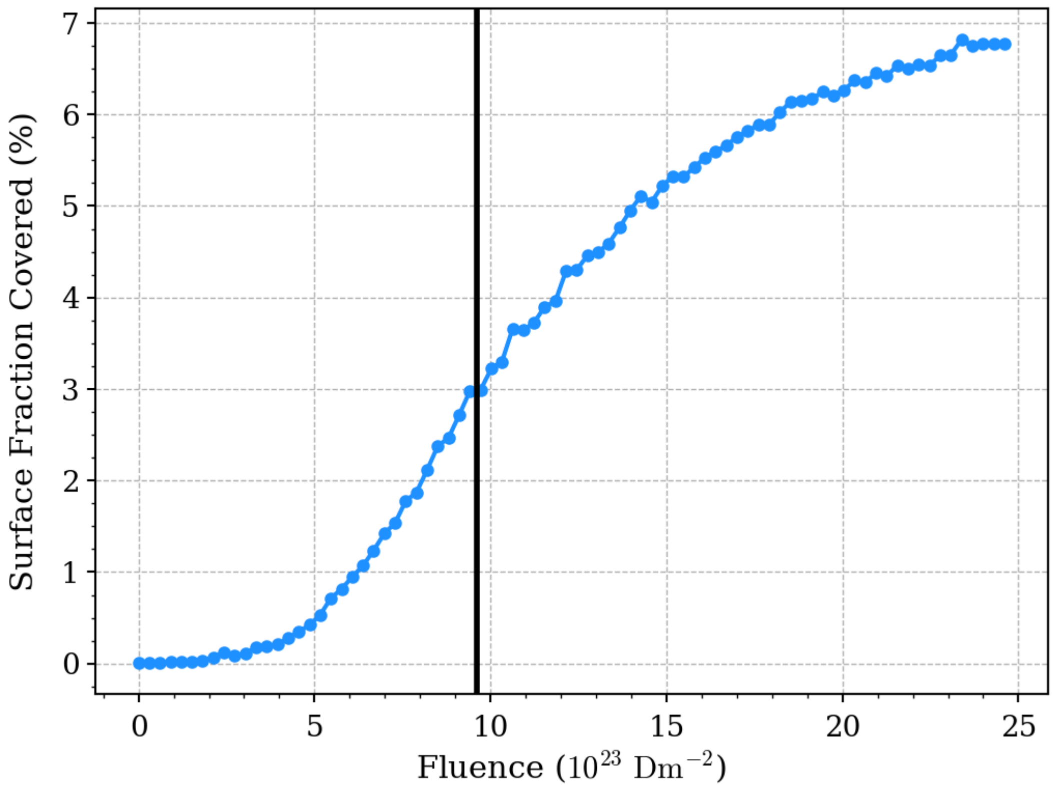


Figure 8.4: *Fraction of the total area that is covered with blisters as function of fluence. As in Fig. 8.3, the black vertical line indicates the fluence that was used as reference for the adaptation of the identification framework. Beyond this value, estimates are becoming less precise.*

sliding window method with a small step size that leads to multiple redundancy. On account of this, the quality of individual detections and the large statistics suffice to give an indication of the general trend at high fluences as well.

A quantity that is more accurately describing the blister formation as a whole is the fraction of the surface that is covered by blister features, since it is to be expected that it is directly related to the amount of hydrogen retained in the specimen, unlike the number of blisters. This quantity is also more robust at high fluences because excess identifications on large features will in aggregate tend more towards the true size of the blister. Hence, this quantity is more likely to adequately represent the overall trend. The progression of the area fraction covered by blisters as function of fluence is depicted in Fig. 8.4. It resembles a similar trend as the number of blisters per area, which was discussed above. At first, there is an accelerating increase before the curve transitions into a phase of steady growth for some time and then begins to saturate. An analogous trend has already been described for similar fluence ranges for blistering on tungsten samples [22]. Compared to the curve for the total number of blisters per area (Fig. 8.3), the increase at low fluences is slower but in turn, the phase of linear growth is longer and the saturation at high fluences progresses slower. A potential explanation is that at the beginning, many small, new blisters occur, while at a later stage already existing blisters continue to expand or smaller blisters merge into larger ones. This interpretation is supported by the analysis of blister size distributions (see further below and in Fig. 8.5). However, in the fluence region where the saturation effect becomes visible, the reliability of the identification process is already reduced to some degree. The saturation effect may therefore be partly attributed to the decreasing identification quality at high fluences. However, this effect is not strong enough to solely account for the saturation at high fluences. Hence, a physical origin of this effect is strongly indicated.

There is a number of plausible mechanisms that may give rise to the observed effect. Firstly, the blister geometry could change over time from flat features at the beginning to more voluminous features later on. This could explain why the rate of increase in the total surface coverage as function of fluence becomes smaller over time. Secondly, as will be discussed further below, blisters may prevent the formation of additional blisters in their close vicinity. This would mean that blisters exhibit an "effective" size that is larger than their apparent size so that the fraction of the total area that is effectively covered would be much larger than estimated. Furthermore, on blisters which cap has already ruptured, no further blisters can emerge since the old blister now constitutes an exhaust channel for the hydrogen. Finally, for larger fluences and correspondingly larger time periods, the hydrogen diffuses deeper into the material and hence accumulates in larger depths. Therefore, the deformations caused by the expansion of cavities occur further below the surface. They may not become recognizable on the surface with the optical system used in the present investigation and thus may not contribute to the total estimated area. However, a dense layer of small blisters close to the surface can suppress the formation of larger blisters further below the surface in the future [116]. A combination of the mechanisms described here is likely to give rise to the observed saturation effect. The tools developed in the present thesis now provide the means by which this effect can be further investigated.

8.2.2 Blister Size Distributions

The size of a blister, i.e. the area it covers on the surface, is one of its most important parameters and therefore an interesting subject of study. Fig. 8.5 depicts the probability density distributions of blister sizes for different initial fluences. The initial fluence is the amount of fluence that the specimen was exposed to before the respective blister occurred. Along with each distribution, a Gaussian kernel density estimate is plotted that has a width equivalent to one third of the standard deviation of the respective histogram. The plot shows that for low fluences, there are a lot of small blisters and just a few medium-sized blisters. However, the larger the total fluence becomes, the broader the respective distribution gets. For high fluences, the number of medium-sized blisters markedly exceeds the number of small blisters. In fact, very large blister features are likely to still be underrepresented because they cannot be estimated correcly by the networks in some cases. Instead, they tend to be split up into multiple smaller detections (cf. Appendix D), which leads to an over-estimation of the number of smaller blisters. Thus, the shift of the distribution peak to larger values for high initial fluences is actually underestimated. This indicates that, for high fluences, already existing blisters grow or merge rather than that new blisters are formed. This interpretation was already discussed above with respect to the saturation effect of the total blister coverage of the surface.

It is noteworthy that the size distributions of blisters depending on the fluence exposure prior to formation have been investigated for tungsten samples by Manhard [22]. He found that his data could be approximated by exponential fits and that the exponential factors of these fits decrease with higher fluence values. Hence, the trend of less small blisters and more large blisters for higher fluences is similar. However, the functional dependencies seem to differ between the two investigation. This could be caused by the difference in material or by differing experimental conditions.[2]

8.2.3 Nearest Neighbor Investigation

As pointed out in Chapter 2 on blister theory, it is not known whether blisters exhibit clustering or structuring behaviour. Also, in Section 8.2.1 it was mentioned that an effective exclusion zone around existing blisters could result in an effective surface coverage that is much larger than the observed coverage. To investigate this, it is appropriate to consider the probability density distribution of the nearest neighbour distances of blisters and compare it to the expected distribution if the positions of the blisters were distributed randomly on the surface. Thereby, it can be determined whether small or large distances are under- or overrepresented. Each of these deviations would indicate either clustering or structuring behavior [117]. For point-like objects that are randomly placed on a two dimensional surface, the number of points in a region of finite size is a random variable with a Poisson distribution. The probability density distribution of nearest neighbour distances for such a Poisson pattern can be found in literature (e.g. [117], Eq. 17) and is given by

$$f(x) = 2\lambda\pi x \exp\left(-\lambda\pi x^2\right) \quad , \tag{8.3}$$

[2]The reader should take note of the fact that Manhard used the diameter of a blister as a measure for its size, whereas here the area covered on the surface was considered. Also, the analysis on tungsten were conducted manually for six different fluence values and separate specimens.

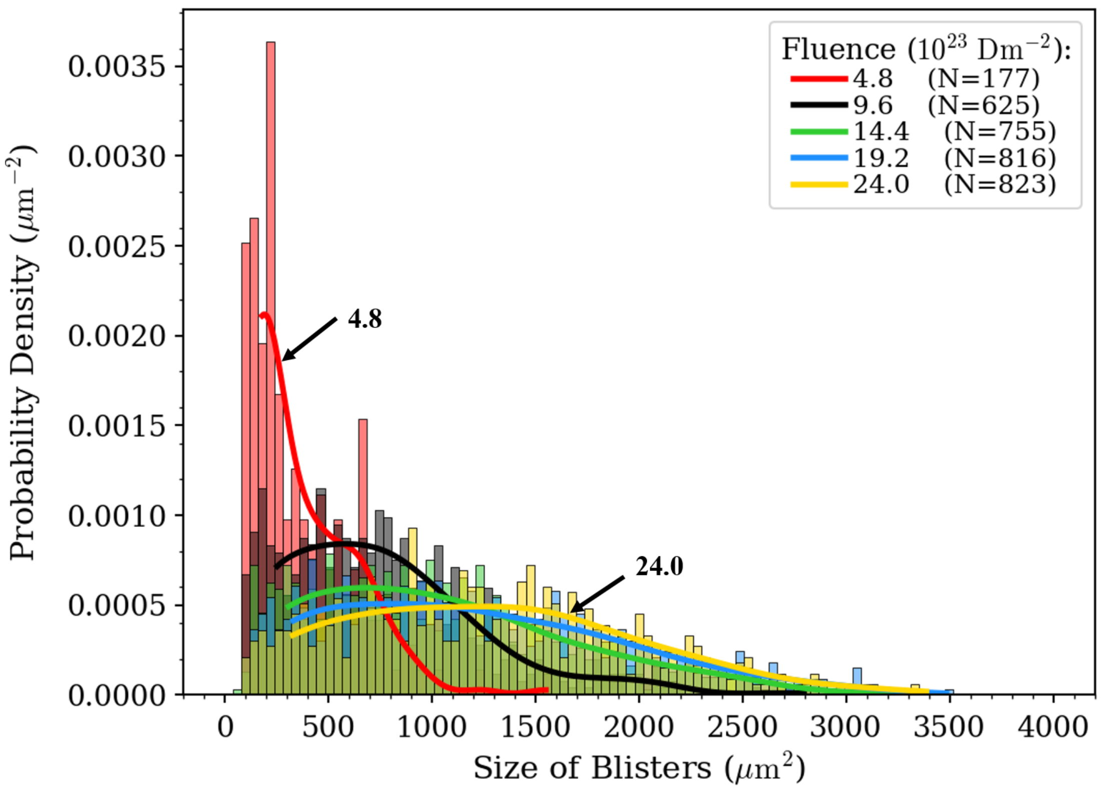


Figure 8.5: *Probability density distributions of the blister sizes for various fluences and with a bin width of 40 μm^2. The size of a blister is defined as the area it covers on the surface. Along with the histograms, a Gaussian kernel density estimate is plotted for each distribution to better illustrate the main trend. For low fluences, there are many small blisters while for higher fluences, the distributions become wider and more flat. It is noteworthy that for very large fluences the number of medium sized blisters exceeds that of small blisters. The width of the distributions at high fluences is likely underestimated due to limitations of the size estimation at the upper and lower end of the distributions that were discussed in Section 8.1.2.*

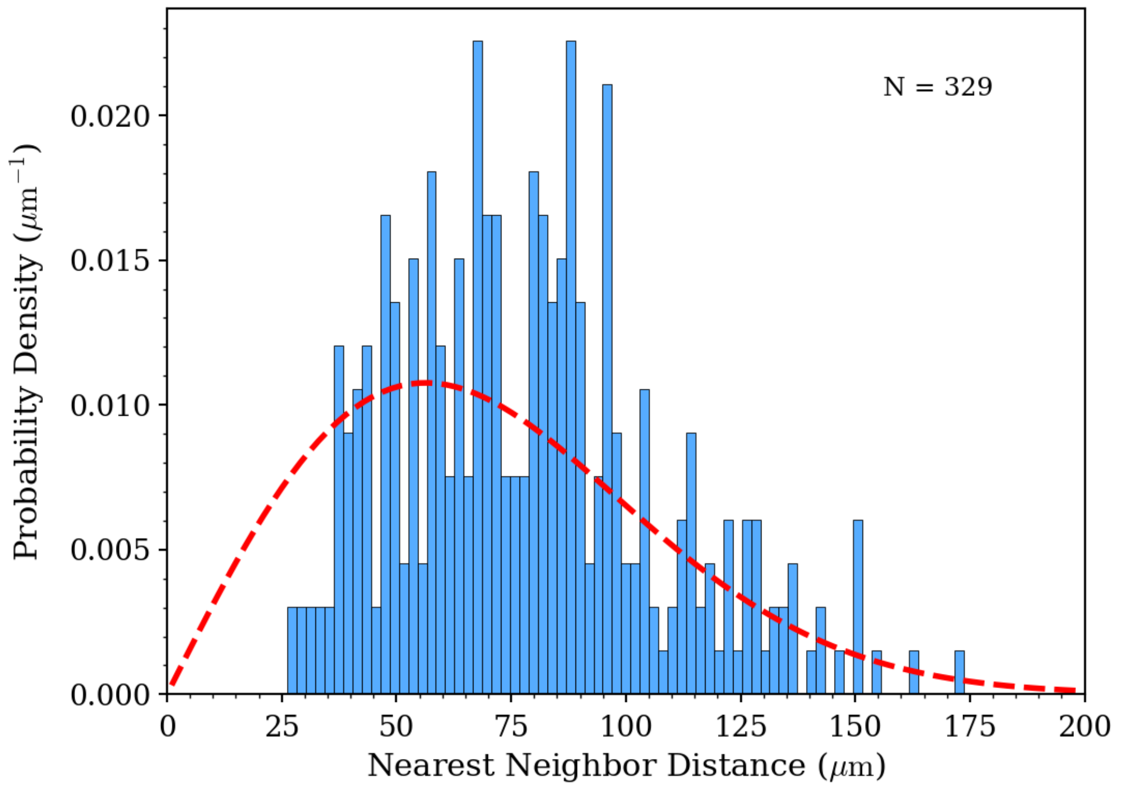


Figure 8.6: *Probability density distribution of the nearest neighbor distances of the blisters in Fig. 8.1b for a fluence of* 9.6×10^{23} Dm^{-2}*. Only blisters from a subregion in the original image contribute to the total statistic of* N *instances because blisters close to the border must be disregarded since their nearest neighbors may lie outside of the considered image section. The red dashed line represents the probability density distribution of the nearest neighbor distances that would follow from an independent and uniform random placement of blisters on the surface, which is given by Eq. (8.3). The discrepancy of both distributions at low nearest neighbor distances is likely caused by inherent limitations of the identification framework, as discussed in the main text. Small distances are certainly not fully excluded, as is discernible from Fig. 8.7. Disregarding this discrepancy, the otherwise good match between the distributions indicates that blisters exhibit neither clustering nor structuring behaviour, at least at medium or large distances.*

where λ is the density of the points on the surface and x is the nearest neighbor distance. For the computation of the nearest neighbor distances in this chapter, only a subregion of the total available area in Fig. 8.1b was selected. Thereby, it is ensured that all nearest neighbors lie inside the region where blisters can actually be detected and boundary effects are excluded. The size of the subregion was about half of the total available area. The probability density distribution of the nearest neighbor distances for the blisters in this subregion is depicted in Fig. 8.6. In addition, the corresponding distribution of the random Poisson pattern for the respective areal blister density is illustrated by the red dashed line. As pointed out, the distributions should match if blisters occur randomly. The figure shows that for large nearest neighbor distances, there is no discernible difference between the random pattern and the actual distribution of the blisters on the surface. For low distances, however, the actual probability density of nearest neighbor distances is significantly lower than that of the Poisson pattern. Instead, there seem to be excess cases for medium distances near the maximum of the distribution. This kind of modification to the distribution corresponds to a Poisson pattern of objects that are not point-like but exhibit an, at least effective, finite size which prevents the positioning of further objects therein. There are two potential causes that could give rise to this effect: Either, small neighboring distances exist but are systematically not recognized by the automated identification framework, or they genuinely do not exist due to some physical reason.

The first possibility comprises cases of misclassifications that occur when blisters are located very close to each other or overlap and one of them does not get identified. An example of such a case is indicated by the arrow in Fig. 8.2c. Here, the actual small neighbor distance does not get registered and larger values are assumed. The minimum neighbor distance that can be identified is limited by both the resolution of the applied optical system (5 μm) and the ability of the identification procedure to distinguish between two blisters that lie closely together, which again depends on the sizes of the blisters. However, the difference between the expected and the actual probability distribution is quite significant and the statistic is quite high (N = 329). Furthermore, the same analysis was conducted across the entire fluence range and the general trend of an underrepresentation of small nearest neighbor distances remained the same. However, from studying a confocal laser scanning image of the sample after the plasma exposure (Fig. 8.7), it can be concluded that small blister distances are at least not completely absent. However, they could still be statistically underrepresented due to a physical effect. This issue could not be clarified with the analysis conducted in the course of the present work and further research on this topic is required.

In conclusion, it can be said that there is no observable long range interaction between blisters that would influence their position on the surface. This is indicated by the good match between the nearest neighbor distributions of the blisters and

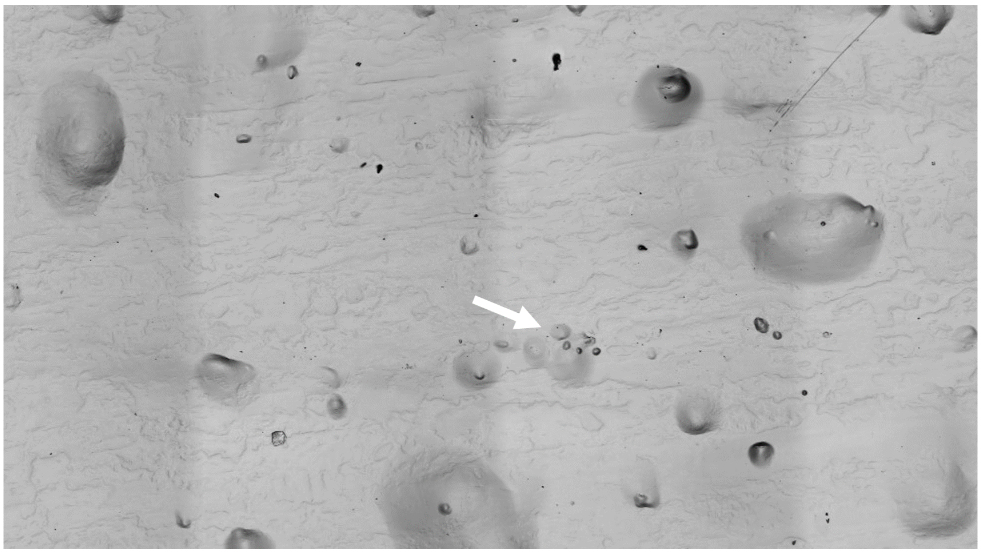


Figure 8.7: *Image of the molybdenum sample investigated in the present thesis after plasma exposure, acquired with a confocal laser scan microscope. The white arrow indicates a region where multiple small blisters are located closely together. This indicates that small nearest neighbor distances between blisters are not completely suppressed.*

the random Poisson pattern for large distances. The validity of the data in this region of the distribution is particularly high since the identification works best for large distances between blisters and hence this conclusion can be regarded as quite certain. However, no assessment can be made with regards to a potential short range interaction between blisters, due to structural limitations of the automated identification procedure for small distances between blisters.

8.3 Results of Time-Dependent Analyses

In this section, the results of the time-dependent analysis will be presented. At first, the quality of the blister tracking is assessed. Then, the gross formation and vanishing rates of blisters are studied. Finally, the distribution of blister lifetimes and the growth dynamics of blisters are investigated.

8.3.1 Quality Assessment of Tracking

In order to conduct a time-dependet analysis, a tracking of the blisters was performed which has been discussed in Section 8.1. Two quantities were considered to assess the quality of the tracking: the mean absolute shift of positions of blisters between successive image cycles and the fraction of blisters that only survived a single image cycle.[3] The respective curves as functions of fluence are depicted in Fig. 8.8. Disregarding the larger statistical fluctuations at low fluences due to a smaller statistical basis, the curves exhibit clear trends. The mean absolute position shift of blisters increases almost linearly with fluence but does not increase above 2.75 pixels and thus is smaller than the optical resolution at all the times. Thus, for those blisters that are tracked correctly, the positional shift is marginal and indicates a good stability of the identification and localization of blisters.

However, the fraction of blisters that were tracked for only one image cycle remains between 0.4 and 0.6 throughout the entire fluence range, which is quite a high value. A certain number of these cases are actual short lived blisters that occurred only for one image cycle, but this fraction could not be determined at this point. However, several undesirable effects may lead to misclassifications with short lifetimes. For example, some of these cases are caused by a somewhat worse alignment of images which causes the false identification of some blisters on the marker that persist only for a single cycle. However, this does not pose a principal problem, as discussed in Section 8.1.2. Furthermore, at high fluences and correspondingly high blister densities, large blister features emerge that consist of either a single large blister or multiple small blisters that overlap. This can cause incorrect or excess identifications on the same feature and leads to frequent label switching and correspondingly short lifetimes. Some examples of this are given in the images in Appendix D. In addition, a significant amount of the one-cycle blisters actually were identified in multiple image cycles but the tracking procedure failed at identifying them as the same blister. Thus, not the identification procedure is failing in these cases but rather the rudimentary matching technique. Therefore, the respective instances were not included in the analysis of blister formation and vanishing rates, which will be discussed in the next subsection. For the other investigations in this section, this is less relevant since only blisters with a minimum lifetime of 20 image cycles are considered, which requires them to be instances with a stable tracking anyway. Additionally, the majority of the remaining blister instances are consistently matched and tracked for more than 30 image cycles, as will be discussed in Section 8.3.3 and can be concluded from Fig. 8.10. This indicates a good matching and tracking of the remaining blister instances and thereby justifies to conduct time-dependent analysis with the present data already. Nonetheless, structural deviations can not be fully excluded since a quantitative typing of blister instances, that are not matched well in successive image cycles, was beyond the scope of the present work. However, a more worthwhile approach to address the label switching issue would be to improve the tracking scheme, which has not been optimized yet.

Overall, there are some difficulties pertaining the accurate tracking of blisters over time, which is caused by a non-sufficing matching scheme and marker artefacts, but

[3]A single image cycle is equivalent to a time period of 5 min and a fluence of 0.3×10^{23} Dm^{-2}.

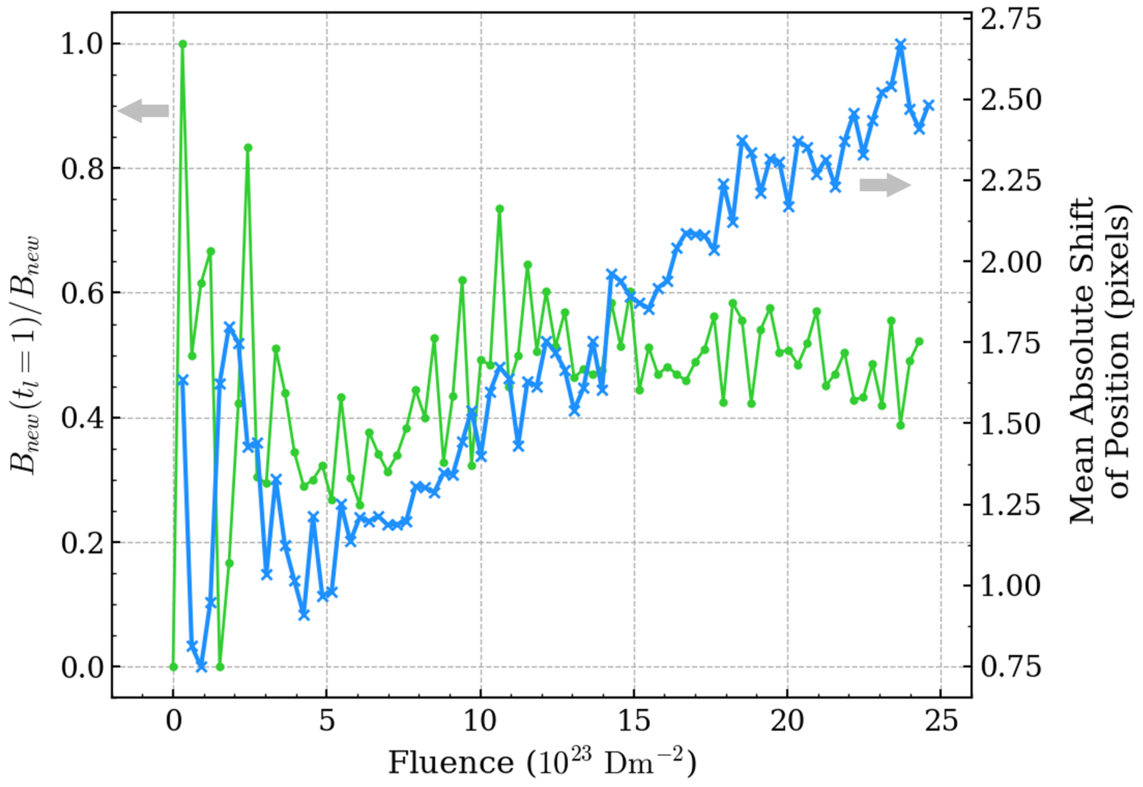


Figure 8.8: *Plot of two quantities that provide information about the quality of the blister tracking. The green line represents the fraction of blisters that were identified in only a single image cycle. The blue line depicts the mean of the absolute position shifts of all blisters between consecutive image cycles, which remains below the optical resolution of 2.76 pixels at all times. The larger fluctuations at small fluences are due to the small number of blisters that contribute to the statistic. The green curve indicates that the matching of the same blister between successive image cycles does not work reliably in all cases. However, the localization of individual blisters is very stable throughout the entire time period.*

the majority of instances gets tracked consistently for a long time and the positional accuracy of blisters that are matched correctly is very good.

8.3.2 Formation and Vanishing Rates of Blisters

Apart from the static analysis that relied on the identification of blisters in single image cycles, some additional information can be gained from the analysis of the dynamic evolution of individual blisters. For example, it yields the gross formation and vanishing rates of blisters and thereby allows to investigate the development of the total number of blisters in more detail. In this analysis, blisters that were tracked only in a single image cycle are disregarded, since some of them may be misclassifications on the marker or large blister features, as discussed above. The plots of all three quantities are given in Fig. 8.9. The figure shows that the formation rate of new blisters increases at the beginning. The vanishing rate of blisters starts to increases somewhat later but catches up at a fluence of about 11×10^{23} Dm^{-2}. From then on, both the formation and vanishing rate increase concurrently so that the total number of blisters begins to saturate. The steady increase in both rates for high fluences is likely caused by a deteriorating quality of the overall identification and tracking results.

8.3.3 Blister Lifetime Distributions

Another interesting question to investigate is whether the lifetimes of blisters depend on the initial fluence, i.e. the fluence the specimen was exposed to before the blisters emerged. For this purpose, the probability density distributions of blister lifetimes for different blister generations, that are grouped by the amount of initial fluence, are given in Fig. 8.10. The different distributions are very similar considering the fact that most instances accumulate at either end of the distribution. At intermediate fluences, only a few instances contribute to the distributions. The consequential uncertainty prevents a detailed analysis in these fluence ranges. To enable this, more image slices need to be analyzed to provide a sufficient statistic. Nevertheless, a general trend can already be recognized in the plot: for higher initial fluences, blisters tend to have shorter estimated lifetimes. The origin of this effect could not be clarified in the course of this analysis. It is not necessarily a physical phenomenon and could instead be caused by the increased difficulty of the identification and tracking of blisters at large fluences as well.

As discussed in Section 8.3.1, a significant amount of the instances that were tracked for only a single image cycle might be erroneous classifications. If the corresponding contribution to the probability distribution is disregarded, the peak at the end of the histogram, which represents the fraction of blisters that lasted for an additional

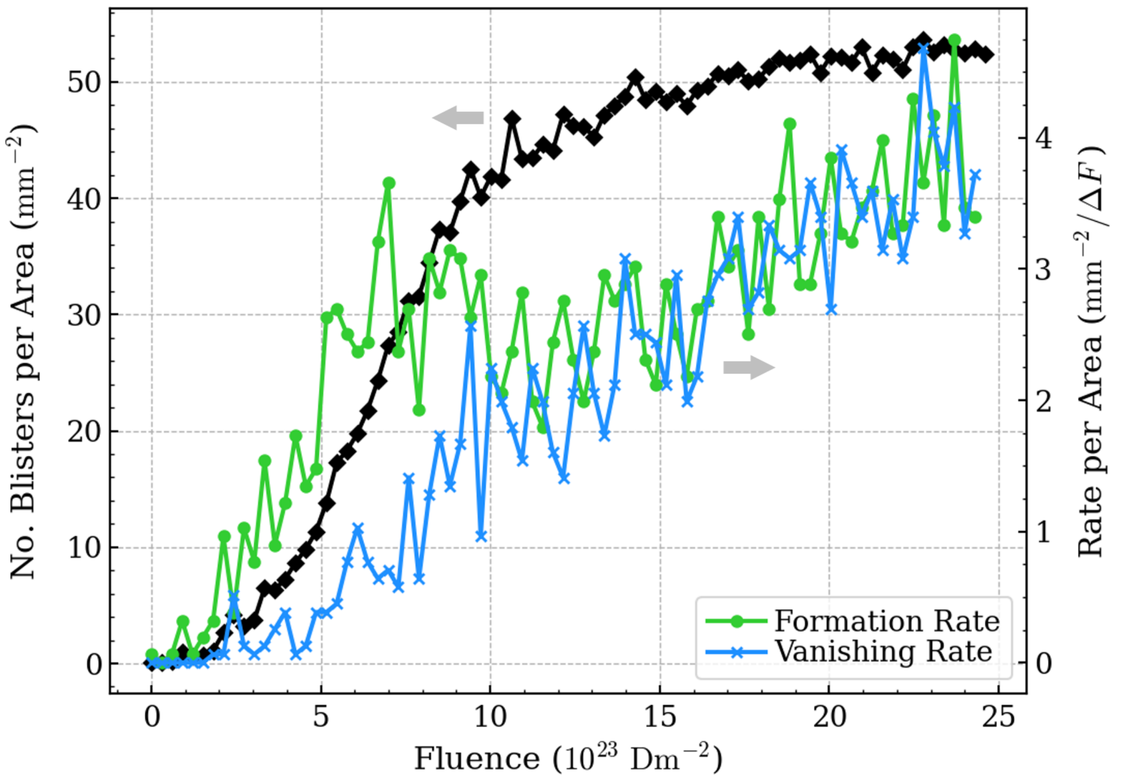


Figure 8.9: *This figure illustrates the development of the number of blisters per area. The black line indicates the total number of blisters per area, which has already been discussed in Section 8.2.1. The green and blue curves denote the formation rate and the vanishing rate of blisters per area as function of fluence. The difference between these curves gives the net change in the total number of blisters per area, i.e. of the black line. Rates are given per image cycle, which is equivalent to a fluence step of* $\Delta F = 0.3 \times 10^{23}$ Dm^{-2}.

fluence of more than 9×10^{23} Dm^{-2} and hence the entire considered time period, typically surpasses the cumulative number of cases of all other fluences. This means that the majority of blisters remains visible on the surface for more than two hours. It also implies that the automated identification routines were able to stably track all of these blisters across the entire time span. This underlines the value that is provided by the investigative approach taken in the present thesis.

8.3.4 Blister Growth Dynamics

Apart from the analysis of blister lifetimes depending on the initial fluences, a similar investigation with regards to the dynamics of blister growth is of interest. For the

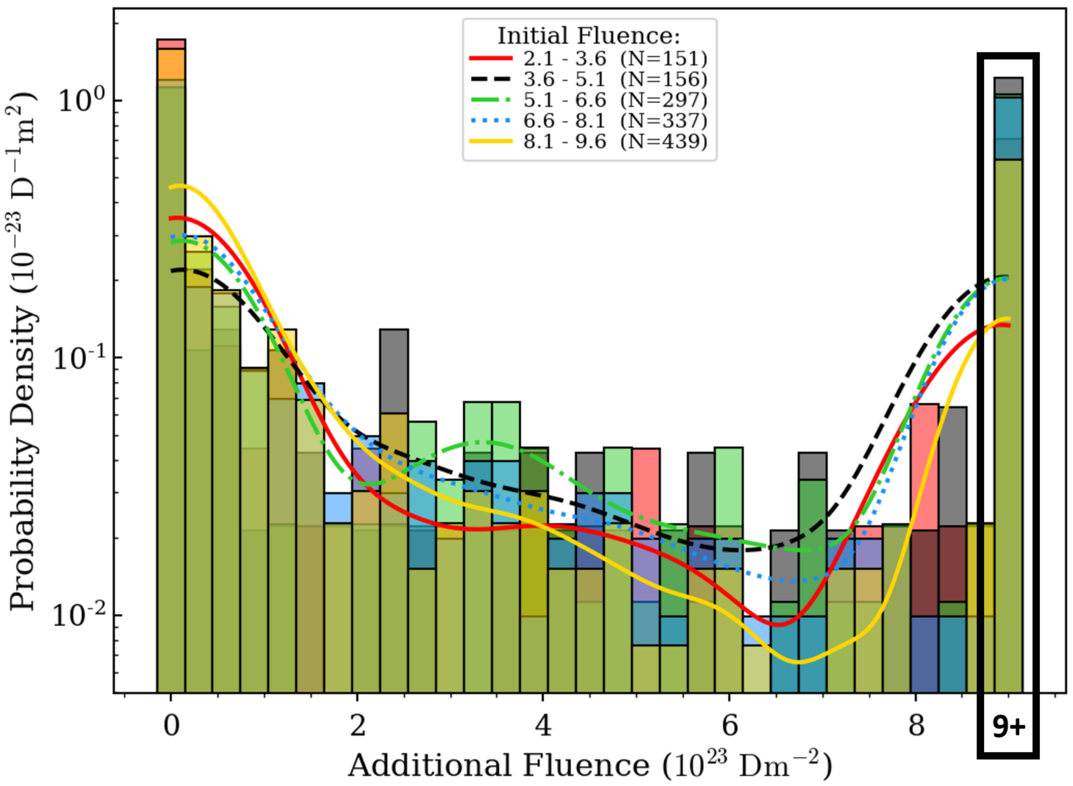


Figure 8.10: *Probability density distribution of blister lifetimes measured by the additional fluence after their emergence. The data is grouped by the initial fluence at the time of emergence of each blister. For each distribution, a Gaussian kernel density estimate is plotted as well. The last bin includes all blisters that lasted for an additional fluence of more than* 9×10^{23} Dm^{-2}*. For each initial fluence range, the corresponding number of identified blisters N is given in the legend.*

comparison of the growth dynamics, the blisters were divided into different generations, i.e. fluence ranges at which they were first detected. Then, from all of these blisters only those were selected that survived for at least 100 minutes, which is equivalent to an additional fluence of 6×10^{23} Dm^{-2}, so that the timelines can be compared to one another. Subsequently, the mean across all remaining blisters in each generation was computed and plotted as function of the additional fluence that the specimen was exposed to (Fig. 8.11). From the figure, it is apparent that blisters that emerge later have a larger size when they are first detected. This could to some degree be caused by label switching on already existing blisters. However, some initial observations of individual blisters have shown that some of the effect seems to be genuine. A quantitative analysis of this effect is therefore required to clarify the different contributions to this phenomenon. The figure further indicates that later

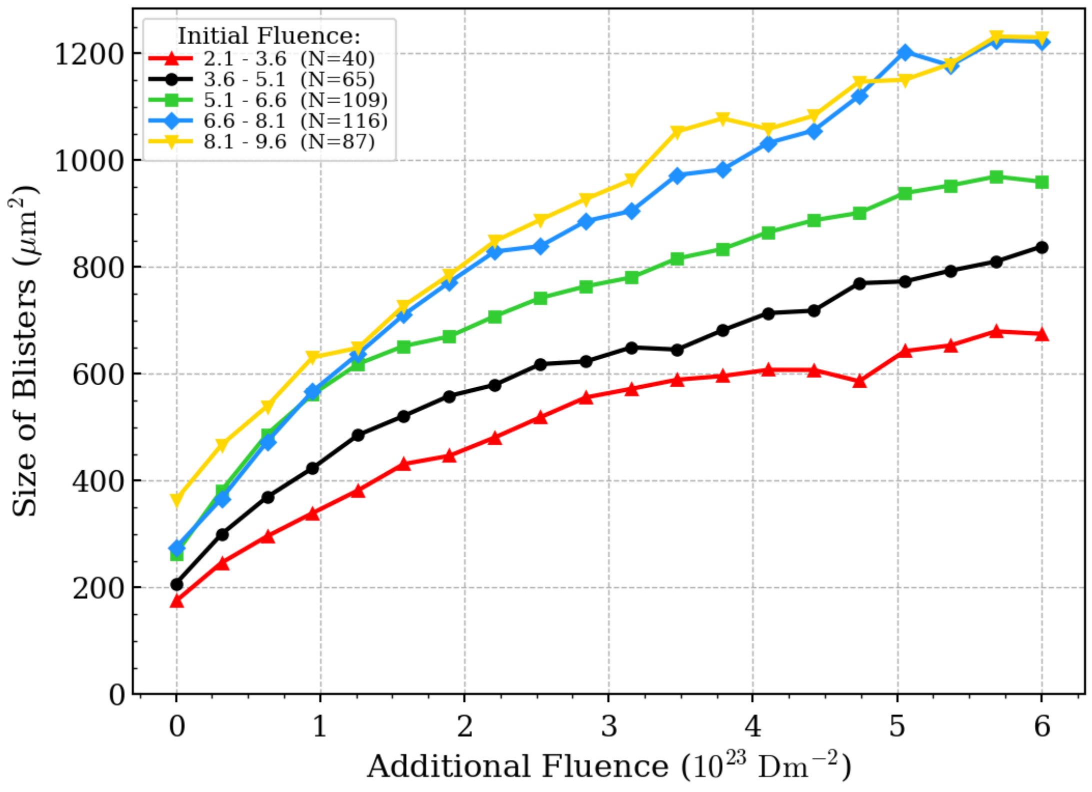


Figure 8.11: *Plot of the development of the mean size of blisters as function of the additional fluence the specimen was exposed to after emergence of each blister. The size of a blister is defined as the area it covers on the surface. The data is grouped by the initial fluence at the time of emergence of each blister. For each group, the number of corresponding instances N is given in the legend as well. Blisters that emerge at higher fluences seem to exhibit larger initial sizes and higher growth rates.*

blister generations exhibit larger growth rates and reach larger sizes after being exposed to the same amount of additional fluence. A potential explanation for this is that later blisters are formed in larger depth, when the hydrogen has diffused further into the bulk. In this case, sufficiently large deformations for detection can only be caused by large structures. Small blisters will not lead to significant protrusions on the surface and hence will not be recognizable. In contrast, small blisters in shallow depth can become visible on the surface so that they can be detected.

A similar analysis was conducted regarding blisters with different lifetimes. For this purpose, blisters from low fluence ranges, where the identification works well, i.e. below a fluence of 9.6×10^{23} Dm^{-2} (cf. Section 8.1.2), were selected and grouped by their lifetime. Then, the average growth dynamic of these blister groups was investigated. The outcome of this analysis did not allow to discern any clear trend. However, this may be attributable to the low statistics (N $\sim$ 40 per group). In order to reach a sufficient number of instances for both of the analysis conducted here, fluence ranges had to be considered where the identification does not work as reliable anymore. Therefore, more image slices need to be analyzed or data from additional samples needs to be accumulated to increase the statistics. This would allow to consider smaller fluence ranges and lifetimes while preserving the statistical significance, which would increase the validity of the interpretations and of the conclusions drawn here.

Chapter 9

Conclusion and Outlook

The present thesis for the first time demonstrated a large-scale and time-resolved in-situ investigation of blistering. This involved the development of a framework for the fully automated identification and characterization of blister features on in-situ acquired image data. To accomplish this, four key issues had to be resolved. First of all, the pronounced texture on the specimen surface impaired the automatic identification of blisters. This was resolved by aligning the images, conducting a background subtraction, and applying advanced denoising techniques, which greatly enhanced the clarity of the blister features in the images. Secondly, obtaining adequate and sufficient training data would have been very laborious if done by hand. Therefore, algorithms were developed that allowed to automatically generate fully parameterized and photo-realistic training data that reproduces the most relevant features of the real data. Thus, a large number of training instances could be provided to the CNNs and hence improve their performance. Furthermore, the correct localization and characterization of multiple blisters in the same image snippet posed a complex task that could not be solved with a single CNN. Therefore, the identification procedure was divided into three distinct subtasks that were addressed with separately trained networks. The choice of appropriate network setups and optimization settings enabled the full automation of the identification procedure while simultaneously achieving both high precision and high reliability of the predictions. Finally, an appropriate strategy for the identification and tracking of blisters was needed. To this end, the CNNs were redundantly applied to the real images via a sliding window scheme and an iterative identification procedure was applied subsequently. Thereby, the reliability of the blister identifications especially for high blister densities could be improved significantly and the tracking of individual blisters over time was enabled.

Together, the resolution of these issues resulted in a reliable and fully automated identification and tracking of individual blisters. This was verified based on the performance on artificially generated images and also validated by applying the identification procedure to real data from a molybdenum sample. This successful model application illustrated the type of investigations that can be performed and the information that can be acquired with this technique and also provided some first insights into the phenomenology of blister formation. In particular, the static analysis of blisters indicated that there are no observable medium- to long-range interactions between blisters that would influence their spatial distribution on the surface. In addition, the time-dependent analysis of blistering indicated that the amount of initial fluence at which blisters emerge influences their growth dynamics while the total lifetime of blisters does not seem to be correlated to these characteristics. Furthermore, no clear relationship between the initial fluence and the distribution of blister lifetimes was discernible with the statistics acquired in the present investigation. A first comparison of these results with a manual evaluation on tungsten conducted by A. Manhard [22] yielded some distinct similarities regarding the abundance and the total surface coverage of blisters and exhibited comparable trends for the evolution of the blister size distributions as function of fluence.

Progressing from here, the technique developed in the present thesis opens up several research opportunities. First of all, it enables the conduction of a comprehensive parameter study that may include variations on experimental parameters such as the energy of the implanted ions and the hydrogen isotopes used, as well as sample parameters like the material and the preparation of the specimen. This would be one of the most promising approaches to advance the knowledge on blister formation and is likely to yield new information on the underlying physical processes. Previous investigations of blisters that involved the manual identification of blister features can also be re-evaluated and improved upon. In addition, the statistical validity of the analysis conducted in the present thesis and the accuracy of the identifications by the developed framework can be improved by utilizing all of the 29 available image slices from the molybdenum sample instead of just one. Furthermore, the framework even allows to incorporate 3D information like the height or the excess volume of blisters in the parameter determination by letting the CNNs infer these quantities implicitly from the side-view images. Confocal laser scan microscopy can be used to acquire 3D-maps from the surface of the specimen for this purpose. The automatic inference of 3D blister information from in-situ image data would constitute a new capability in the investigation of blistering that does not exist so far. Beyond the present application, the developed framework may also be adapted to address other tasks as well, as long as appropriate training data can be created.

If required, the identification procedure of blisters can be improved in several ways. For example, it would be useful to diversify the training data for the CNNs by adding blisters with more complex shapes and increasing the range of some scenic parameters to specifically address certain types of misclassifications. This is likely to improve the precision especially on large blisters, since they tend to exhibit more irregular shapes. Another possibility is to provide the CNNs with some additional training instances consisting of real images that were labeled by hand. Thereby, the performance on irregular blister features is likely improved. Furthermore, it would be an interesting approach to use "committees" of CNNs [49] for the identification of blisters. In this case, the same task is performed by several CNNs with different structures and the different evaluations are then used to make better decisions. For example, features in the images that were not represented well by the training data are likely to give rise to diverging results and can hence be excluded more reliably than with a single network alone. In addition, this divergence of predictions could be used as a quality measure for individual identifications.

Overall, the present thesis addresses several fundamental issues that hampered the research on blistering phenomena so far, in particular

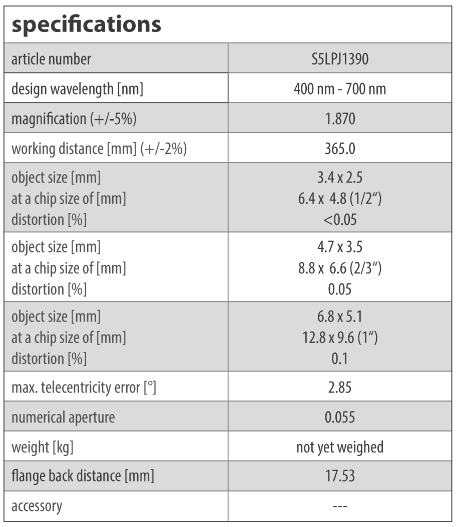
- the huge manual effort that was needed to analyze image data from blistered surfaces and
- the difficulty to ensure the traceability of the identification process and the consistency of blister identifications across different applications when evaluated manually.

In the course of this, a comprehensive framework for the automatic identification and characterization of blisters in image data was developed that features good reliability and high precision in particular for low blister densities while being transparent and reproducible at the same time. This framework now allows to conduct quantitative and time-resolved studies of blister formation thoroughly and in a multitude of experimental scenarios. It can thereby contribute to a better understanding of blistering phenomena and the underlying physics.

Appendix

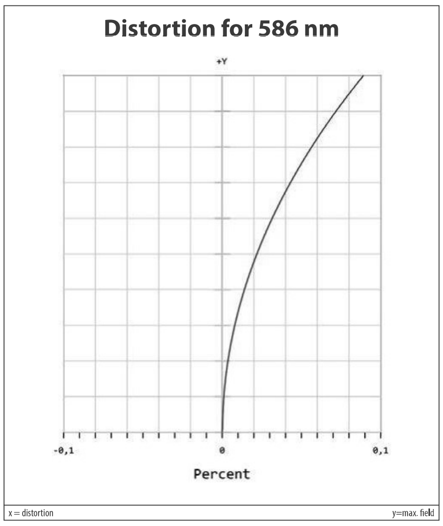
A Data Sheets of the Employed Optical System

These are the data sheets of the optical system that was used to acquire the images of the Mo sample.

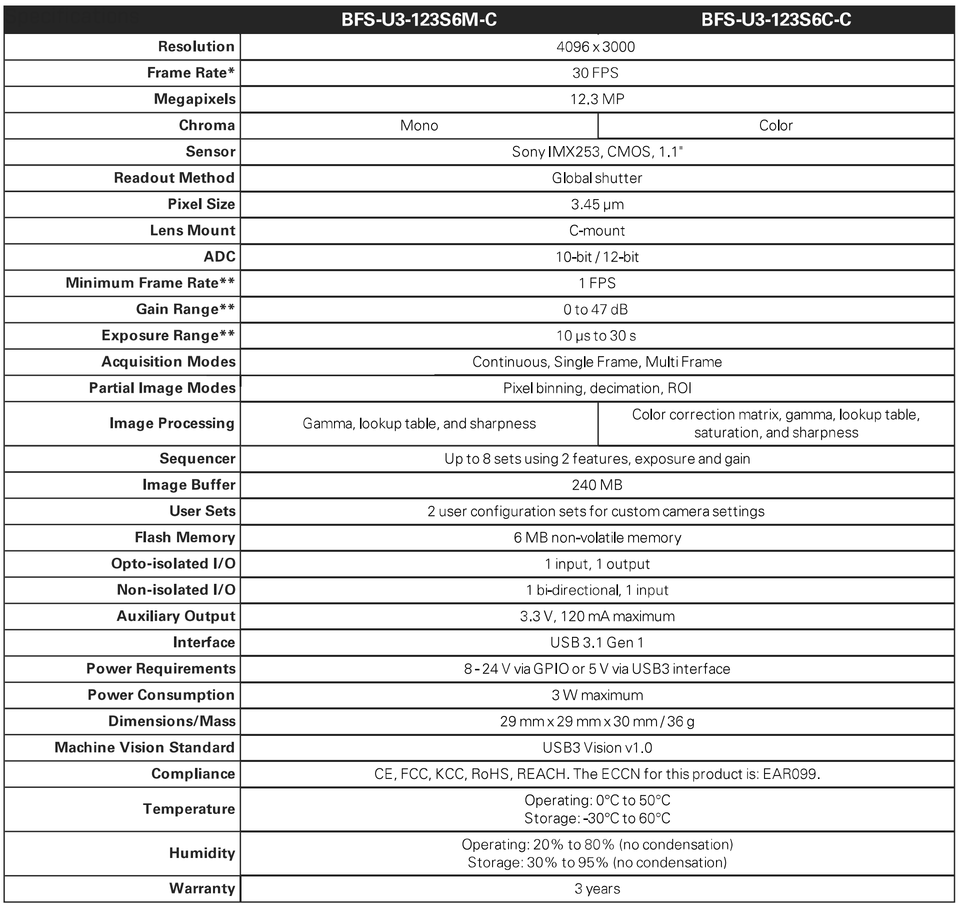


specifications

article number	S5LPJ1390
design wavelength [nm]	400 nm - 700 nm
magnification (+/-5%)	1.870
working distance [mm] (+/-2%)	365.0
object size [mm] at a chip size of [mm] distortion [%]	3.4 x 2.5 6.4 x 4.8 (1/2") <0.05
object size [mm] at a chip size of [mm] distortion [%]	4.7 x 3.5 8.8 x 6.6 (2/3") 0.05
object size [mm] at a chip size of [mm] distortion [%]	6.8 x 5.1 12.8 x 9.6 (1") 0.1
max. telecentricity error [°]	2.85
numerical aperture	0.055
weight [kg]	not yet weighed
flange back distance [mm]	17.53
accessory	---



Data sheet of the camera.



	BFS-U3-123S6M-C	BFS-U3-123S6C-C
Resolution	4096 x 3000	
Frame Rate*	30 FPS	
Megapixels	12.3 MP	
Chroma	Mono	Color
Sensor	Sony IMX253, CMOS, 1.1"	
Readout Method	Global shutter	
Pixel Size	3.45 µm	
Lens Mount	C-mount	
ADC	10-bit / 12-bit	
Minimum Frame Rate**	1 FPS	
Gain Range**	0 to 47 dB	
Exposure Range**	10 µs to 30 s	
Acquisition Modes	Continuous, Single Frame, Multi Frame	
Partial Image Modes	Pixel binning, decimation, ROI	
Image Processing	Gamma, lookup table, and sharpness	Color correction matrix, gamma, lookup table, saturation, and sharpness
Sequencer	Up to 8 sets using 2 features, exposure and gain	
Image Buffer	240 MB	
User Sets	2 user configuration sets for custom camera settings	
Flash Memory	6 MB non-volatile memory	
Opto-isolated I/O	1 input, 1 output	
Non-isolated I/O	1 bi-directional, 1 input	
Auxiliary Output	3.3 V, 120 mA maximum	
Interface	USB 3.1 Gen 1	
Power Requirements	8 - 24 V via GPIO or 5 V via USB3 interface	
Power Consumption	3 W maximum	
Dimensions/Mass	29 mm x 29 mm x 30 mm / 36 g	
Machine Vision Standard	USB3 Vision v1.0	
Compliance	CE, FCC, KCC, RoHS, REACH. The ECCN for this product is: EAR099.	
Temperature	Operating: 0°C to 50°C Storage: -30°C to 60°C	
Humidity	Operating: 20% to 80% (no condensation) Storage: 30% to 95% (no condensation)	
Warranty	3 years	

Data sheet of the telecentric lens.

B Code Listings for the Creation of Artificial Training Data

```python
import random
import os
import pickle
import math
from configs_and_resources import *

# choose SCENARIOS to run
SCENARIOS = [111, 211, 212, 213, 214, 314]

# set DATA DIRECTORY
os.chdir(r'C:\Users\aroessel\Desktop\training_data')

# set NUMBER OF IMAGES to be created
N_IMAGES = 17000 # 128^2 = 16384

# set OUTPUT PRECISION
N_PARAM = 7 # number of OUTPUT PARAMETERS per blister
p_out = 4 # precision of output

# set IMAGE PARAMETERS
AREA_WIDTH = 217
AREA_HEIGHT = 638

# set GLOBAL BLISTER PARAMETERS
R_MIN = 15
R_MAX = 40

# loop over all SCENARIOS
for scenario in SCENARIOS:
    # create output directory
    OUT_DIR = f'scenario-{scenario}'
    if not os.path.exists(OUT_DIR):
        os.mkdir(OUT_DIR)

    # initialize OUTPUT ARRAY
    labels = np.zeros((N_IMAGES, N_MAX_BLISTER, N_PARAM))

    # LOOP for desired NUMBER OF IMAGES
    for image_instance in range(N_IMAGES):

        # set OVERALL SETTINGS
        ## choose NUMBER OF BLISTERS for this specific image
        if scenario in [101, 111]:
            n_blisters = random.randint(0, N_MAX_BLISTER)
        elif scenario in [201, 211]:
            n_blisters = 1
        elif scenario in [202, 212]:
            n_blisters = 2
        elif scenario in [203, 213]:
            n_blisters = 3
        elif scenario in [204, 205, 214]:
            n_blisters = 4
        elif scenario in [303, 304, 314]:
            n_blisters = random.randint(1, N_MAX_BLISTER)
        print(image_instance, "    n_blister: ", n_blisters)

        ## choose LIGHTING SETTINGS
        light_pos_x = -100000 * random.random()
        light_pos_z = 82800 + 10430 * random.random()

        ## choose TEXTURE SETTINGS
        texture_randomizer = 100000 * random.random()
        texture_intensity = 0.1

        # prepare OUTPUT STRING for POVRAY
        out_pr_environment = f"""
            light_source {{
              <{light_pos_x}, 270000, {light_pos_z}>
              color rgb <2.2, 2.2, 2.2>
            }}
            #declare body_texture_plane =
              texture {{
                pigment {{ color rgb 1 }}
                normal {{marble {texture_intensity} scale <2000,300,100> turbulence<1, 0.001, 1000> omega 0.55 frequency 3.2}}
                finish{{
                  diffuse 0.3
                  ambient 0.03
                  specular 0.2
                  reflection {{0.2 metallic}}
                }}
                translate<0,{texture_randomizer},0>
              }}
            """

        # initialize OUTPUT VARIABLES
        out_pr_blister = f""

        # LOOP for the NUMBER OF BLISTERS in this specific image
        blister_positions = []
        for j in range(n_blisters):
            # loop until valid blister position was found
            while True:
                # set SPECIFIC BLISTER PARAMETERS (size; height; position; scaling; rotation)
                radius = R_MIN + (R_MAX - R_MIN) * random.triangular(0, 1, 0.25)
                radius = round(radius, p_out)

                z = - 0.5 * radius - 0.43 * radius * random.triangular(0, 1, 0.75)
                z = round(z, p_out)

                scale_x = round(1 + 1.5 * random.triangular(0, 1, 0.25), p_out)
                scale_y = round(1 + 1.5 * random.triangular(0, 1, 0.25), p_out)
                scale_z = 1

                rot_angle = round(90 * random.random(), p_out)

                # set absolute POSITION OF BLISTER in image ((0,0) in top left corner)
                if scenario in [304, 314] and j == 0:
                    x_out = np.random.normal(AREA_WIDTH/2, AREA_WIDTH * 2/64)
                    y_out = np.random.normal(AREA_HEIGHT/2, AREA_HEIGHT * 2/64)
                else:
                    x_out = AREA_WIDTH * random.random()
                    y_out = AREA_HEIGHT * random.random()

                # calculate position of blister in povray coordinates ((0,0) in middle)
                x = x_out - AREA_WIDTH / 2
                x = round(x, p_out)

                y = y_out - AREA_HEIGHT / 2
                y = - y  # this is necessary because in the image, the y-vector is negative
                y = round(y, p_out)

                # ensure that blisters are not too close to each other
                if j == 0:
                    blister_positions.append([x, y])
                    break

                distances = []
                for pos in blister_positions:
                    dist = np.sqrt((x - pos[0])**2 + (y - pos[1])**2)
                    distances.append(dist)

                min_dist = np.amin(np.array(distances))

                if min_dist > radius:
                    blister_positions.append([x, y])
                    break

            # write blister parameters to povray output
            out_pr_blister += f'sphere{{ <0, 0, 0>, {radius}  \n          scale<{scale_x}, {
    scale_y}, {scale_z}> \n        rotate<0, 0, {rot_angle} > \n          translate<{x}, {y}, {z}
    > \n     texture{{body_texture}} }} \n \n'

            # prepare labels for output
            width = 2. * scale_x * math.sqrt(radius ** 2 - z ** 2)
            length = 2. * scale_y * math.sqrt(radius ** 2 - z ** 2)
            height = round(radius + z, p_out) # note: height is not normalized and can be quite a
     large number

            ## normalize position and width in image to (0,1)
            x_out = round(x_out / AREA_WIDTH, p_out)
            y_out = round(y_out / AREA_HEIGHT, p_out)
            width = round(width / AREA_WIDTH, p_out)
            length = round(length / AREA_HEIGHT, p_out)
            area = round( (np.pi * width / 2 * length / 2) * (WINDOW_SHAPE[0] * WINDOW_SHAPE[1] *
     2.94) , p_out) # the second factor is the total number of pixels of the total visible area
    when viewed from the top, 2.94 is the factor introduced by the angle of view

            # assign labels to output array
            labels[image_instance, j, :] = [1, x_out, y_out, area, rot_angle, width, length]

        # save povray input file
        povray_input = open(f'current_blister.inc', 'w')
        povray_input.write(out_pr_environment + out_pr_blister)
        povray_input.close()

        # run povray render for current image_instance
        os.system(fr'pvengine64 /render process settings.ini Output_File_Name=\\e2mgast2019a\
    Users\aroessel\Desktop\training_data\scenario-{scenario}\{str(image_instance).zfill(6)}.png /
    EXIT')

    # PRINT final array of LABELS for this scenario
    print(f"\n Result of Scenario {scenario}:")
    print(labels, "\n")

    # SAVE LABELS to directory
    pickle_out = open(OUT_DIR + "\labels.pkl", 'wb')
    pickle.dump(labels, pickle_out)
    pickle_out.close()
```

Listing 1: *This script generates the artificial training data that is used to train the CNNs by randomly setting the required parameters and executing a corresponding POV-Ray script, which is given below. The POV-Ray execution settings are provided in the third code listing below.*

```
#include "colors.inc"
global_settings {assumed_gamma 1.0}
#declare body_texture =
  texture {
    pigment { color rgb 1 }
    //normal {marble 0.3 scale <2000,300,20> turbulence<1, 0.001, 1000> omega 0.55 frequency 3.2}
    finish{
      diffuse 0.3
      ambient 0.03
      specular 0.2
      reflection {0.2 metallic}
    }
  }
// 1 unit = 1 micrometer
camera {
  orthographic
  sky z
  up <0, 1, 0>
  right <-1, 0, 0>
  location  <0, -365000, 131700>
  look_at   <0.0, 0, 0>
  angle 0.032 //for 64p width
}
#include "current_blister.inc"
box {
    <-4000, -4000, -8>, <4000, 4000, 0>
    texture {body_texture_plane}
}
```

Listing 2: *This is the POV-Ray script that actually generates the artificial images according to the settings given in the code listing above.*

```
Width = 64
Height = 64
Bits_Per_Color = 8  ; sets bit depth to x bits per color (default: 8)
Output_File_Type = N  ; sets the Output File Type to x (N=PNG) (default: 24-bit BMP)
```

Listing 3: *These are the execution settings for the POV-Ray script above.*

C Code Listings for the Setup and Training of the CNNs

```python
'''
NN configs: Use this to adapt the search scope for the NNs. These code snippets are shared among all three NNs. If seperate trials are needed for the NNs, just copy the appropriate code to the respective NN file and make the appropriate changes.
'''

# the following imports are (partially) necessary for the NN codes themselves to function and are imported via this module implicitly
import tensorflow as tf
from keras import regularizers
from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D, Input, Concatenate
from kerastuner.tuners import BayesianOptimization
import numpy as np
import os
import cv2
import pickle
import time
from tensorflow.keras import Model

# Number of images used for training
N_IMAGES = 16384

# def image import function
def load_images(DATADIR):
    X = []
    for img in os.listdir(DATADIR)[:N_IMAGES]:
        img_array = cv2.imread(os.path.join(DATADIR, img), cv2.IMREAD_GRAYSCALE)
        X.append(img_array)
    X = np.array(X)
    X = X / 255.
    X = X.reshape(X.shape[0], X.shape[1], X.shape[2], 1) # this reshape is necessary to match the Tensorflow format
    return(X)


# set hyperparameters for model stream and tuner search.
def add_model_stream(hp, X):
    # initialize layers dictionary and last layer
    layers = {}
    layers['last_layer'] = None

    # create input layer
    layers['input_img'] = Input(shape=X.shape[1:], name='input_img')

    # choose hyperparameters
    regularization = regularizers.l1(0)

    # add first convolution and MaxPooling layer to model
    conv1_kernel_size = 7
    conv1_stride = hp.Int(f"conv1_strides", min_value=1, max_value=2, step=1)
    layers['conv1'] = Conv2D(filters=hp.Int('conv1_n-filters', min_value=32, max_value=96, step=32),
                             kernel_size=(conv1_kernel_size, conv1_kernel_size),
                             strides=(conv1_stride, conv1_stride),
                             input_shape=X.shape[1:],
                             activation="relu",
                             padding="same",
                             kernel_regularizer=regularization)(layers['input_img'])

    layers['maxPool1'] = MaxPooling2D(pool_size=(2, 2))(layers['conv1'])
    layers['last_layer'] = layers['maxPool1']

    # add additional convolution and some MaxPooling layers to model
    n_conv = hp.Int("n-add-conv", 8, 16, 1)
    maxPools = []
    for mp in range(hp.Int("n-add-MaxPool", 0, 4, 1)):
        maxPools.append(hp.Int(f"MaxPool_{mp + 2}", min_value=2, max_value=n_conv + 1, step=1))

    n_filter1 = hp.Int(f"conv_2-6_n-filters", min_value=32, max_value=192, step=32)
    n_filter2 = hp.Int(f"conv_7-end_n-filters", min_value=32, max_value=192, step=32)

    for i in range(2, n_conv + 2, 1):
        if i < 7:
            layers[f'conv{i}'] = Conv2D(n_filter1, kernel_size=(3, 3), strides=(1, 1), activation="relu",
                                        padding="same", kernel_regularizer=regularization)(layers['last_layer'])
            layers['last_layer'] = layers[f'conv{i}']
        else:
            layers[f'conv{i}'] = Conv2D(n_filter2, kernel_size=(3, 3), strides=(1, 1), activation="relu",
                                        padding="same", kernel_regularizer=regularization)(layers['last_layer'])
            layers['last_layer'] = layers[f'conv{i}']

        if i in maxPools:
            layers[f'maxPool{i}'] = MaxPooling2D(pool_size=(2, 2))(layers['last_layer'])
            layers['last_layer'] = layers[f'maxPool{i}']

    layers['flatten'] = Flatten()(layers['last_layer'])
    layers['last_layer'] = layers['flatten']

    # add dense layers to model
    for i in range(1, hp.Int("n-dense", 1, 3, 1) + 1, 1):
        layers[f'dense{i}'] = Dense(hp.Int(f"dense{i}_n-units", min_value=64, max_value=384,step=64),
                                    activation="relu",
                                    kernel_regularizer=regularization)(layers['last_layer'])
        layers['last_layer'] = layers[f'dense{i}']

    layers['dropout'] = Dropout(hp.Float('dropout', min_value=0.1, max_value=0.5, step=0.1))(layers['last_layer'])
    layers['last_layer'] = layers['dropout']

    return(layers, regularization)


def conduct_search(build_model, LOGDIR, X, y):
    # define tuner instance
    tuner = BayesianOptimization(
        build_model, objective="val_loss", max_trials=64, num_initial_points=16, executions_per_trial=1, directory=LOGDIR)

    # conduct search
    tuner.search(X, y, batch_size=64, epochs=128, validation_split=0.125)
    return(tuner)
```

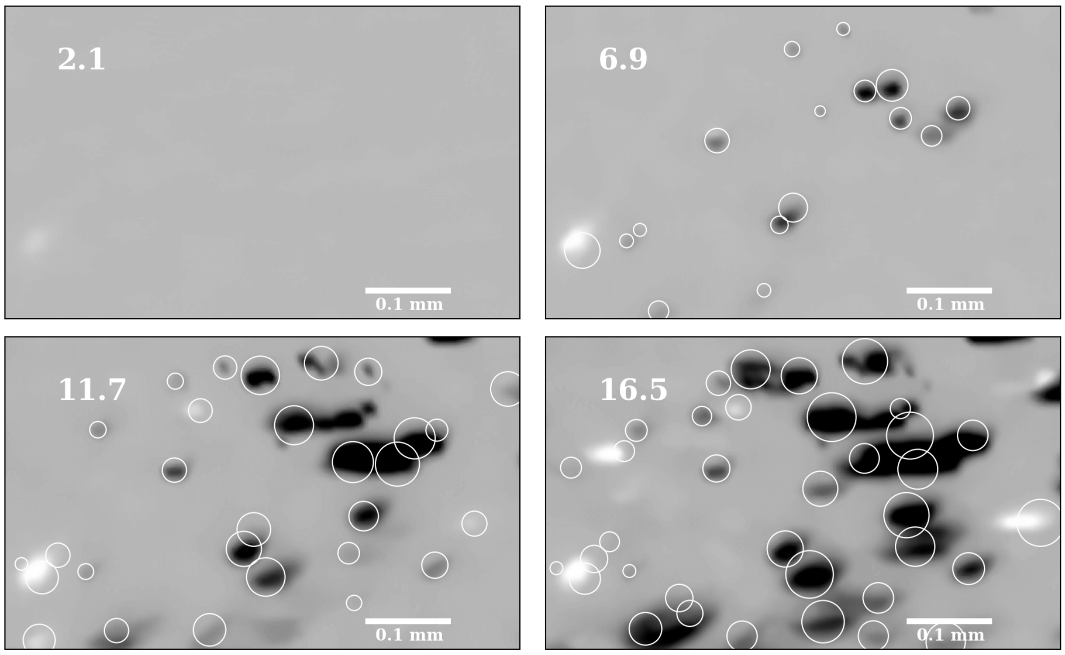
Listing 4: *This file manages the setup of the CNNs. It includes all settings and prerequisites that are common to the training and optimization of all employed CNNs, regardless of their specific task. This file will be imported in the scripts that actually train the different networks. The next code listing below will give a sample of these specific scripts.*

```python
from NN_configs import *

# choose scenario
for SCENARIO in [214]:

    # set directories
    time_stamp = time.strftime('%Y-%m-%d_%H.%M.%S')
    DATADIR = fr"\\e2mgast2019a\Users\aroessel\Desktop\training_data\scenario-{SCENARIO}"
    LOGDIR = fr"C:\Users\aroessel\Desktop\models\tuner\2.2_localization\scenario-{SCENARIO}\{time_stamp}"
    MODELDIR = fr"\\e2mgast2019a\Users\aroessel\Desktop\models\tuner\2.2_localization\scenario-{SCENARIO}\best_models\{time_stamp}"

    if not os.path.exists(LOGDIR):
        os.mkdir(LOGDIR)

    ## configure things for different number classifications
    # set number of blisters to be identified
    if SCENARIO in [201, 211]:
        N_BLISTER = 1
    elif SCENARIO in [202, 212]:
        N_BLISTER = 2
    elif SCENARIO in [203, 213]:
        N_BLISTER = 3
    elif SCENARIO in [204, 214, 205, 206]:
        N_BLISTER = 4

    # prepare list of permutations
    permutations = np.array([
        [3, 2, 1, 0],[3, 2, 0, 1],[3, 1, 2, 0],[3, 1, 0, 2],[3, 0, 2, 1],[3, 0, 1, 2],[2, 3, 1, 0],[2, 3, 0, 1],[2, 1, 3, 0],[2, 1, 0, 3],[2, 0, 3, 1],[2, 0, 1, 3],[1, 3, 2, 0],[1, 3, 0, 2],[1, 2, 3, 0],[1, 2, 0, 3],[1, 0, 3, 2],[1, 0, 2, 3],[0, 3, 2, 1],[0, 3, 1, 2],[0, 2, 3, 1],[0, 2, 1, 3],[0, 1, 3, 2],[0, 1, 2, 3]
    ])
    if SCENARIO in [201, 211]:
        permutations = np.expand_dims(permutations[0, -1:], 1)
    elif SCENARIO in [202, 212]:
        permutations = permutations[0:2, -2:]
    elif SCENARIO in [203, 213]:
        permutations = permutations[0:6, -3:]
    elif SCENARIO in [204, 214, 205, 206]:
        pass
    n_perm = len(permutations)

    # load images
    X = load_images(DATADIR)

    # load labels
    y = pickle.load(open(os.path.join(DATADIR, "labels.pkl"), "rb"))
    y = y[:N_IMAGES, :N_BLISTER, (1, 2)]
    y = y.reshape(y.shape[0], -1)

    # define custom loss functions TODO: (must this be normalized?)
    def my_loss(y_true, y_pred):
        for index in range(n_perm):
            permutation = permutations[index]
            d_tot = np.zeros(64)
            for i in range(N_BLISTER):
                d_blister = np.zeros(64)
                for j in range(2):
                    dist = (y_pred[:, 2 * permutation[i] + j] - y_true[:, 2 * i + j]) ** 2
                    d_blister += dist
                d_tot += tf.math.sqrt(d_blister)
            d_tot = tf.expand_dims(d_tot, 1)
            if index == 0:
                output = d_tot
            else:
                output = tf.concat([output, d_tot], 1)
        output = tf.math.reduce_min(output, axis=1)
        output = tf.math.reduce_mean(output)
        return output

    def build_model(hp):

        layers, regularization = add_model_stream(hp, X)

        # create output layer
        layers['output'] = Dense(2 * N_BLISTER, activation="sigmoid", name="output", kernel_regularizer=regularization)(layers['last_layer'])

        # compile model
        model = Model(inputs=[layers['input_img']], outputs=[layers['output']])
        model.compile(loss=my_loss, optimizer="adam", metrics=["accuracy"])
        model.optimizer.learning_rate = hp.Choice('l_rate', [1e-4, 2e-4, 5e-4, 7e-5, 4e-5, 2e-5])

        return model

    # conduct search
    tuner = conduct_search(build_model, LOGDIR, X, y)

    # activate this to save the best model for later reloading
    best_model = tuner.get_best_models()[0]
    best_model.save(MODELDIR)

    # save tuner logs
    with open(LOGDIR + ".pkl", "wb") as f:
        pickle.dump(tuner, f)

    # output results of best trials
    print(tuner.results_summary())
```

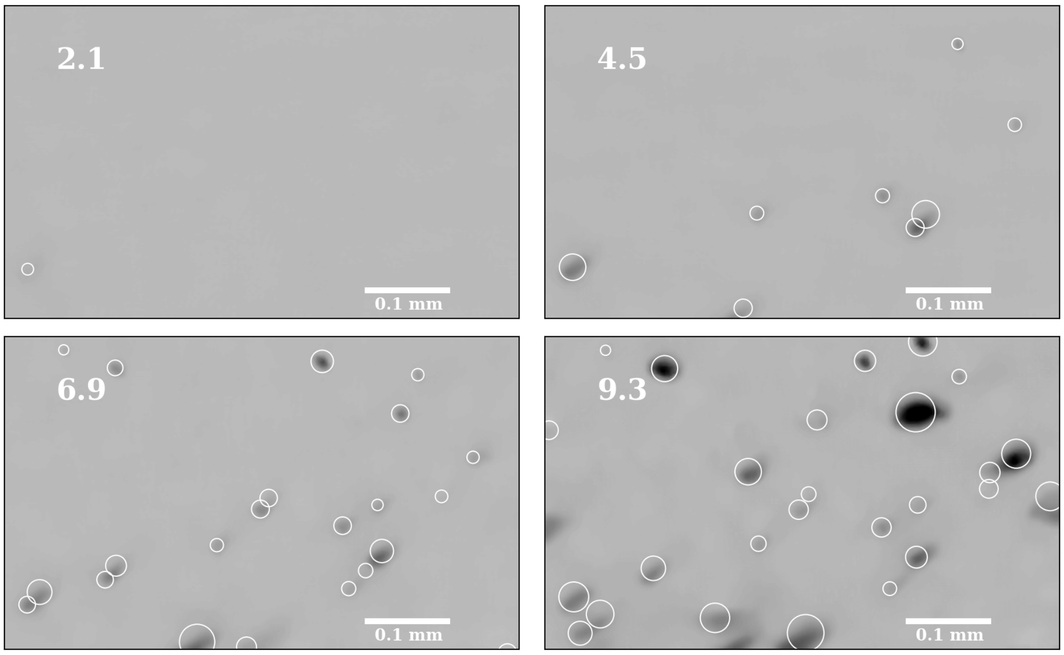
Listing 5: *This script is an example of how the networks are trained and optimized. In this case, it is responsible for localizing a specific number of blisters in a* 64×64 *image snippet. The codes for the other network types are very similar and only differ with respect to output layers and their activation functions as described in the main part of this thesis, which is why they are not given here. All common specifications are imported from a common configuration file, which is given above.*

D Sample Images of Detection Results on Real Data

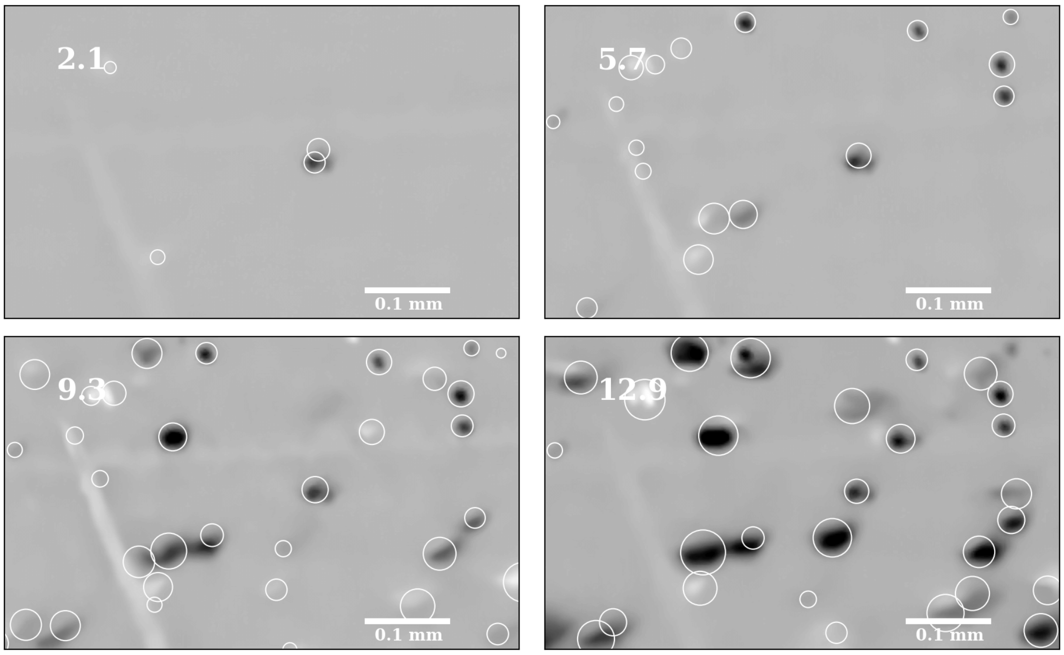
These are sample images that illustrate the identification of blisters on real images. Each of the following nine figures I - IX consists of four consecutive images of the same region on the surface of the specimen but for different fluence values. The fluence is given in the top left corner of each image in units of 10^{23} Dm^{-2}. The images are superimposed with the blister identifications, which are indicated by white circles. The center of the circle is located at the estimated position and the area of the circle is chosen such that it is identical to the estimated area of the blister as if the surface of the specimen was viewed from the top. The training data was adapted to blister features at a fluence of 9.6×10^{23} Dm^{-2}, so beyond this, the detection quality is expected to deteriorate significantly.



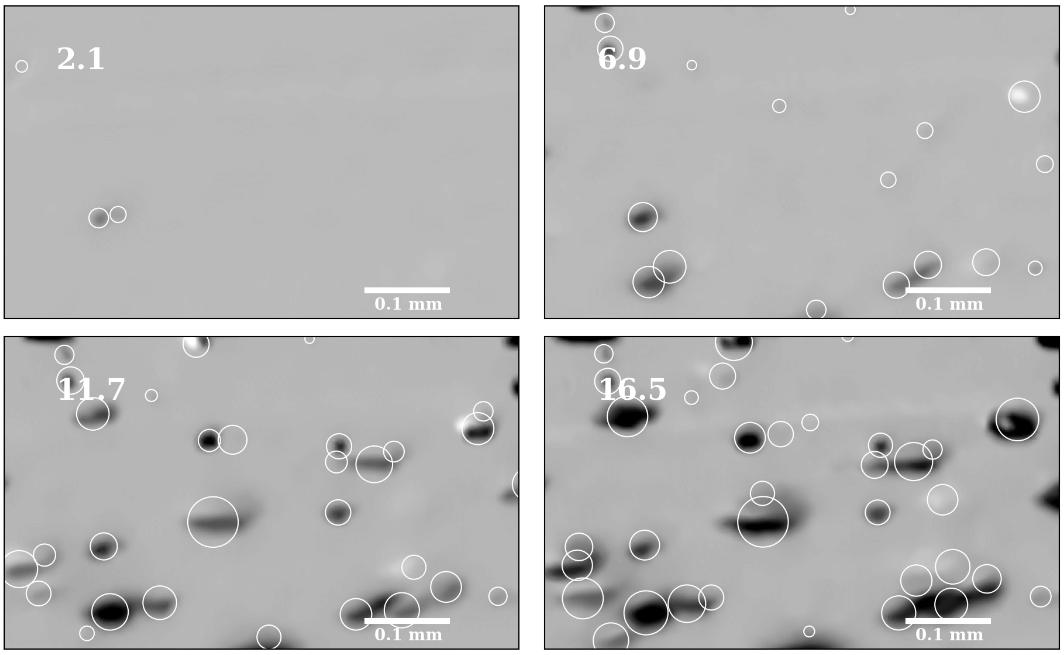
I



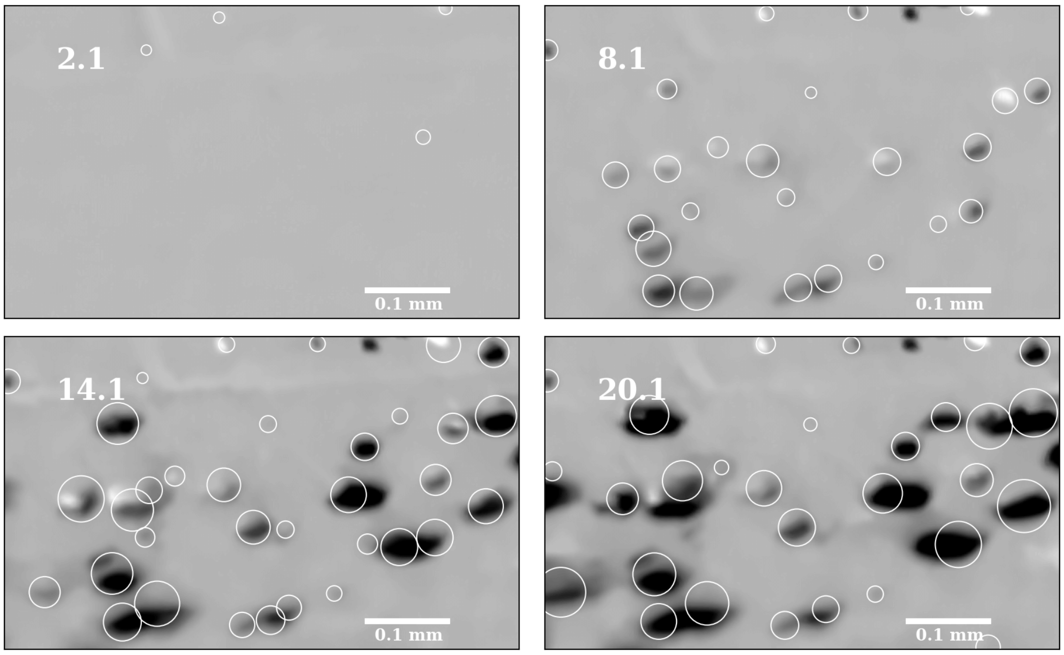
II



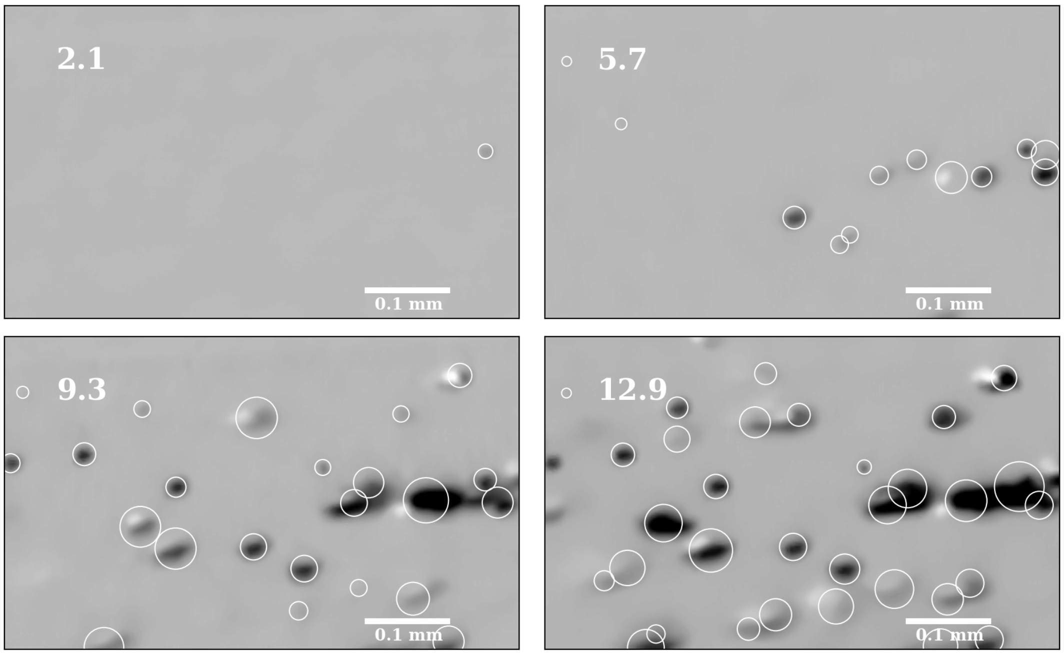
III


2.1
0.1 mm
6.9
0.1 mm
11.7
0.1 mm
16.5
0.1 mm

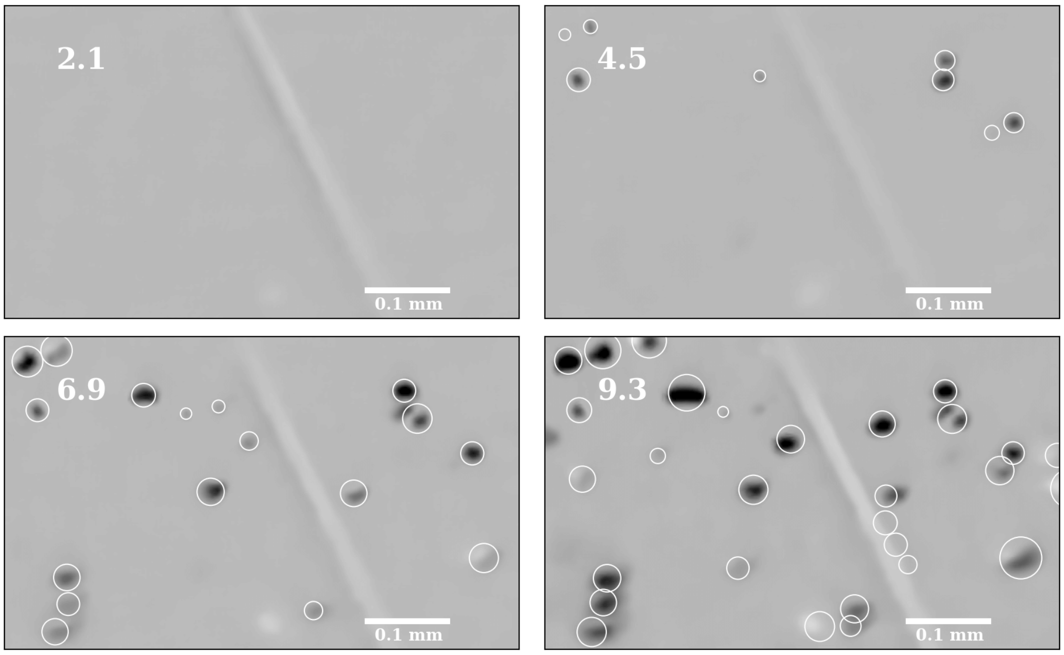
IV


2.1
0.1 mm
8.1
0.1 mm
14.1
0.1 mm
20.1
0.1 mm

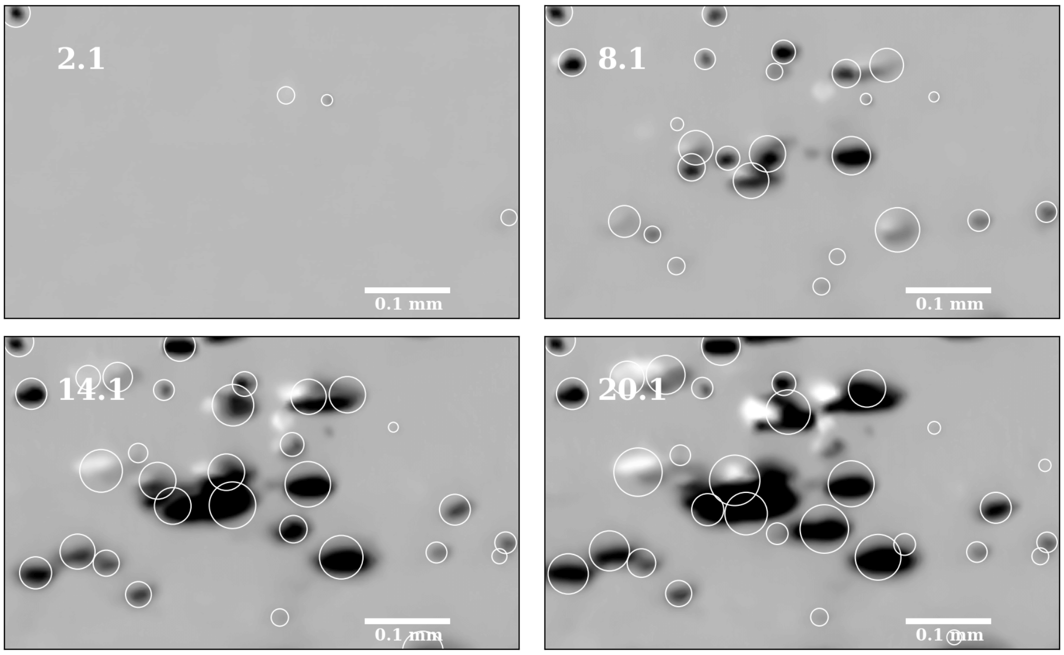
V



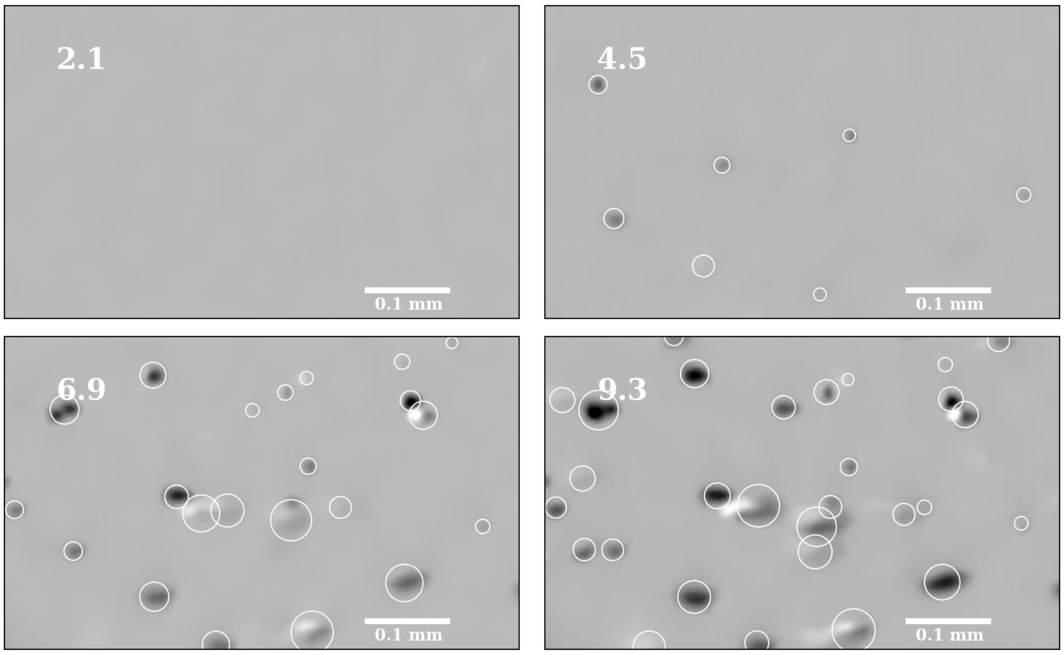
VI



VII


2.1
8.1
14.1
20.1
0.1 mm
0.1 mm
0.1 mm
0.1 mm

VIII


2.1
4.5
6.9
9.3
0.1 mm
0.1 mm
0.1 mm
0.1 mm

IX

E Code Listings for the Analysis of Real Data

```
'''
This file contains general configurations for training and analysis as well as backend resources
'''

import numpy as np

# choose slice to be analyzed
SLICE = 21   # done: 4, 14, 21
trial_id = 3    # choice between different analysis scenarios

# choose parameter to be determined
PARAMETER = 3   # 3 = area, other options: rotation angle, x axis, y axis

# define constants
## shape of sliding window and inputs for NNs
WINDOW_SHAPE = (64, 64)

## maximum number of blisters in window
N_MAX_BLISTER = 4

# choose training data scenarios to be used for training
DATADIR_ART_N = fr"\\e2mgast2019a\Users\aroessel\Desktop\training_data\scenario-{111}"
DATADIR_ART_LOC_1 = fr"\\e2mgast2019a\Users\aroessel\Desktop\training_data\scenario-{211}"
DATADIR_ART_LOC_2 = fr"\\e2mgast2019a\Users\aroessel\Desktop\training_data\scenario-{212}"
DATADIR_ART_LOC_3 = fr"\\e2mgast2019a\Users\aroessel\Desktop\training_data\scenario-{213}"
DATADIR_ART_LOC_4 = fr"\\e2mgast2019a\Users\aroessel\Desktop\training_data\scenario-{214}"
DATADIR_ART_PARAM = fr"\\e2mgast2019a\Users\aroessel\Desktop\training_data\scenario-{314}"


# choose specific models to be used for analysis
## for number classification
MODEL_DIR_N = fr"C:\Users\aroessel\Desktop\models\tuner\2.1_number-classification\scenario-111\
      best_models\2020-09-29_09.07.04"

## for localization
MODEL_DIR_LOC_1 = fr"C:\Users\aroessel\Desktop\models\tuner\2.2_localization\scenario-211\
      best_models\2020-08-29_20.44.02"
MODEL_DIR_LOC_2 = fr"C:\Users\aroessel\Desktop\models\tuner\2.2_localization\scenario-212\
      best_models\2020-08-30_15.31.39"
MODEL_DIR_LOC_3 = fr"C:\Users\aroessel\Desktop\models\tuner\2.2_localization\scenario-213\
      best_models\2020-08-31_09.14.18"
MODEL_DIR_LOC_4 = fr"C:\Users\aroessel\Desktop\models\tuner\2.2_localization\scenario-214\
      best_models\2020-09-30_11.27.56"

## for parameter determination
MODEL_DIR_PARAM = fr"C:\Users\aroessel\Desktop\models\tuner\2.3_parameter-determination\scenario
      -314\best_models\2020-09-01_18.47.18"

'''
additional backend resources:
'''
class Scene:
    """
    Scene is equivalent to one slice. It contains the images of all cycles and the information of
    all blisters that were identified during analysis.
    """
    def __init__(self, shape, n_cycles, images):
        self.shape = shape
        self.n_cycles = n_cycles
        self.max_ID = 0

        # initialize variables and assign values
        # self.cycles = []
        self.images = images  # array of images of all cycles
        self.blisters = {}
        self.deleted_blisters = None

    def add_Blister(self, cycle, position, n_classifications, area, sds):
        """
        """
        next_ID = self.max_ID + 1
        self.blisters[f'{next_ID}'] = Blister(self.n_cycles, cycle, position, n_classifications,
     area, sds, next_ID)
        self.max_ID = next_ID
        return(fr"Added blister with key {next_ID}")

    def get_max_ID(self):
        '''
        :returns the currently highest blister index
        '''
        return(self.max_ID)

```

```python
    def reset(self):
        self.blisters = {}
        self.max_ID = 0

    def get_Blisters_in_ROI(self, xy, sh):
        """
        :returns the keys of all blisters in a certain region
        xy = roi_start
        sh = roi_shape
        """
        output = []
        for key in self.blisters.keys():
            cycle = self.blisters[key].start_cycle
            pos = self.blisters[key].positions[cycle, :]
            if int(pos[0]) in range(xy[1], xy[1] + sh[1]) and int(pos[1]) in range(xy[0], xy[0] + sh[0]):
                output.append(key)
        return(output)


class Cycle:
    '''
    This class is meant for static analysis without tracking over time.
    It has similar features as the Scene class.
    '''
    def __init__(self, shape, image):
        self.shape = shape
        self.image = image
        self.max_ID = 0
        self.blisters = {}

    def add_Blister(self, position, n_classifications, area, sds):
        next_ID = self.max_ID + 1
        self.blisters[f'{next_ID}'] = Blister(1, 0, position, n_classifications, area, sds, next_ID)
        self.max_ID = next_ID
        return(fr"Added blister with key {next_ID}")

    def get_max_ID(self):
        return(self.max_ID)


class Blister:
    """
    This class contains all blister parameters and features that are relevant during analysis.
    """
    def __init__(self, n_cycles, cycle, position, n_classifications, area, sds, key):
        '''
        This method initializes a blister with specific values at the appropriate time (=cycle).
        '''
        self.start_cycle = cycle
        self.final_cycle = cycle
        self.key = str(key)

        # initialize time series and add values
        self.positions = np.zeros((n_cycles, 2))
        self.shifts = np.zeros((n_cycles, 2))
        self.n_classifications = np.zeros(n_cycles)
        self.areas = np.zeros(n_cycles)
        self.sds = np.zeros((n_cycles, 2))

        self.positions[cycle, :] = position
        self.n_classifications[cycle] = n_classifications
        self.areas[cycle] = area
        self.sds[cycle, :] = sds

    def add_cycle(self, cycle, position, n_classifications, area, sds):
        '''
        This method updates a blister by adding all information in a additional cycle.
        '''
        self.positions[cycle, :] = position
        self.n_classifications[cycle] = n_classifications
        self.areas[cycle] = area
        self.sds[cycle, :] = sds
        self.shifts[cycle, :] = position - self.positions[cycle - 1, :]

        self.final_cycle = cycle

    def get_lifetime(self):
        return(self.final_cycle - self.start_cycle)

    def is_active(self, cycle):
        if self.n_classifications[cycle] == 0:
            return(False)
        else:
            return(True)

class Prediction:
    """
    Used for static analysis of single cycles. It contains the initial predictions made by
    the networks before blister identification.
    """
    def __init__(self, grid_numbers, grid_locations, abs_locations, pixel_locations):
        self.grid_numbers = grid_numbers
        self.grid_locations = grid_locations
        self.abs_locations = abs_locations
        self.pixel_locations = pixel_locations


def analyse_roi(roi_list_tmp):
    """
    Returns the mean position, standard deviation and number of localizations in a
    region of interest. Takes a list of blister localizations in (x, y) format!
    Calculations are performed via first and second moments.
    Output is x and y coordinates in pixels in the side-view
    """
    roi_list = np.array(roi_list_tmp)
    n = roi_list.shape[0]

    vec_x = roi_list[:, 0]
    x_first = np.sum(vec_x) / n
    x_second = np.sum(vec_x ** 2) / n
    x_sd = np.sqrt(np.maximum(x_second - x_first ** 2, 0.))

    vec_y = roi_list[:, 1]
    y_first = np.sum(vec_y) / n
    y_second = np.sum(vec_y ** 2) / n
    y_sd = np.sqrt(np.maximum(y_second - y_first ** 2, 0.))

    output = [x_first, y_first, n, x_sd, y_sd]
    return(output)
```

Listing 6: *This is a configuration file that aids the identification and analysis of blisters in the following scripts. Most importantly, it defines a class 'Blister' that contains all information that was acquired for a specific blister instance.*

```
from configs_and_resources import *
import os
import cv2
import numpy as np
import pickle as pkl

# set parameters
n_cycles = 100

# set directories
IMAGE_DIR = fr"C:\Users\aroessel\Desktop\real_data\04_denoised\slice-{str(SLICE).zfill(3)}"
TARGET_DIR = fr"C:\Users\aroessel\Desktop\real_data\05_analyses\slice-{str(SLICE).zfill(3)}\trial
    -{str(trial_id).zfill(3)}"

if not os.path.exists(TARGET_DIR):
    os.mkdir(TARGET_DIR)
    os.mkdir(TARGET_DIR + "\cycle_predictions")
    os.mkdir(TARGET_DIR + "\cycle_identifications")
    os.mkdir(TARGET_DIR + "\calculations")
    os.mkdir(TARGET_DIR + "\dynamic_analysis")
    os.mkdir(TARGET_DIR + "\plots")

# load images
images = []
for img in os.listdir(IMAGE_DIR)[:n_cycles]:
    img_array = cv2.imread(os.path.join(IMAGE_DIR, img), cv2.IMREAD_GRAYSCALE)
    images.append(img_array)
images = np.array(images)
images = images/255.
if trial_id in [1, 2]:
    images = images[:, 88:344, 3032:]
else:
    pass

if trial_id in [1, 2]:
    shape = (256, 1024)
else:
    shape = images[0].shape

```

```
# initialize slice
slice = Scene(shape, n_cycles, images)

# write file to directory
with open(TARGET_DIR + fr"\scene.pkl", 'wb') as f: # _{time.strftime('%Y-%m-%d_%H.%M.%S')}
    pkl.dump(slice, f)
```

Listing 7: *This script prepares a 'scene' instance and loads the image data.*

```
from tensorflow import keras
from configs_and_resources import *
from NN_configs import *

# set parameters
## stepsizes for sliding window steps
if trial_id in [1]:
    SLIDING_STEPS = (16, 16)
else:
    SLIDING_STEPS = (8, 8)

# load scene instance for this slice:
DATADIR = fr"C:\Users\aroessel\Desktop\real_data\05_analyses\slice-{str(SLICE).zfill(3)}\trial-{
    str(trial_id).zfill(3)}\scene.pkl"
with open(DATADIR, 'rb') as f:
    scene = pickle.load(f)

# set and create output directory
OUT_DIR = fr"C:\Users\aroessel\Desktop\real_data\05_analyses\slice-{str(SLICE).zfill(3)}\trial-{
    str(trial_id).zfill(3)}\cycle_predictions"
if not os.path.exists(OUT_DIR):
    os.mkdir(OUT_DIR)

# choose cycles to be analyzed
CYCLES = []
for i in range(0, 100):
    CYCLES.append(i)

# load and compile models
model_n = keras.models.load_model(MODEL_DIR_N)

model_loc_1 = keras.models.load_model(MODEL_DIR_LOC_1, compile=False)
model_loc_2 = keras.models.load_model(MODEL_DIR_LOC_2, compile=False)
model_loc_3 = keras.models.load_model(MODEL_DIR_LOC_3, compile=False)
model_loc_4 = keras.models.load_model(MODEL_DIR_LOC_4, compile=False)

def my_loss():
    pass

model_loc_1.compile(loss=my_loss, optimizer="adam", metrics=["accuracy"])
model_loc_2.compile(loss=my_loss, optimizer="adam", metrics=["accuracy"])
model_loc_3.compile(loss=my_loss, optimizer="adam", metrics=["accuracy"])
model_loc_4.compile(loss=my_loss, optimizer="adam", metrics=["accuracy"])

# calculate number of full steps possible in y and x direction
## + 1 is necessary to account for 0th step in sliding window
N_STEPS = ((scene.shape[0] - WINDOW_SHAPE[0]) // SLIDING_STEPS[0] + 1, (scene.shape[1] -
    WINDOW_SHAPE[1]) // SLIDING_STEPS[1] + 1)

for CYCLE in CYCLES:
    print("\n", fr"Started with Cycle: {CYCLE}")

    # load image
    image = scene.images[CYCLE].copy()

    # prepare output data structures (grid means the grid spanned by the sliding window step size
    )
    grid_numbers = np.zeros((N_STEPS[0], N_STEPS[1], N_MAX_BLISTER + 1))
    grid_locations = np.zeros((N_STEPS[0], N_STEPS[1], N_MAX_BLISTER, 4))
    abs_locations = []
    pixel_locations = np.zeros(scene.shape, dtype=np.int16)

    # MAKE PREDICTIONS
    for i in range(N_STEPS[0]):
        for j in range(N_STEPS[1]):
            # get image section
            y_start = 0 + i * SLIDING_STEPS[0]
            y_end = y_start + WINDOW_SHAPE[0]
            x_start = 0 + j * SLIDING_STEPS[1]
            x_end = x_start + WINDOW_SHAPE[1]

            image_section = image[y_start:y_end, x_start:x_end]
            image_section_tf = np.expand_dims(image_section, (0, 3))  # this converts the image to tensorflow format

            # PREDICT NUMBER (number of blisters in each analyzed frame)
            grid_numbers[i, j, :] = model_n.predict(image_section_tf)

            # PREDICT LOCATIONS
            pred_loc = []
            if np.argmax(grid_numbers[i, j, :]) == 1:
                pred_loc.append(model_loc_1.predict(image_section_tf))
            elif np.argmax(grid_numbers[i, j, :]) == 2:
                pred_loc.append(model_loc_2.predict(image_section_tf))
            elif np.argmax(grid_numbers[i, j, :]) == 3:
                pred_loc.append(model_loc_3.predict(image_section_tf))
            elif np.argmax(grid_numbers[i, j, :]) == 4:
                pred_loc.append(model_loc_4.predict(image_section_tf))

            pred_loc = np.array(pred_loc)
            pred_loc = np.reshape(pred_loc, (-1, 2))

            for b in range(pred_loc.shape[0]):
                # save relative positions
                grid_locations[i, j, b] = (1, pred_loc[b, 0], pred_loc[b, 1], 0)

                # save absolute positions
                abs_x = x_start + pred_loc[b, 0] * (WINDOW_SHAPE[1] - 1)
                abs_y = y_start + pred_loc[b, 1] * (WINDOW_SHAPE[0] - 1)
                abs_locations.append([1, abs_x, abs_y])

                # add predicted location to pixel-wise positions frame
                pixel_x = int(round(abs_x, 0))
                pixel_y = int(round(abs_y, 0))
                pixel_locations[pixel_y, pixel_x] += 1

        print(i)
    print("finished Cycle: ", CYCLE)

    abs_locations = np.array(abs_locations)

    # write OUTPUT to disk
    output = Prediction(grid_numbers, grid_locations, abs_locations, pixel_locations)

    with open(OUT_DIR + f"\cycle-{str(CYCLE).zfill(3)}.pkl", 'wb') as f:
        pickle.dump(output, f)#, protocol=pickle.HIGHEST_PROTOCOL)
```

Listing 8: *This script is responsible for the implementation of the sliding window approach and the application of the CNNs onto the image data. It results in the 'heat map' of detections as described in the main text.*

```
from tensorflow import keras
import pickle
from configs_and_resources import *

# choose image to be analyzed
CYCLES = []
for i in range(0, 100):
    CYCLES.append(i)

# set general parameters for analysis
if trial_id in [1]:
    n_class_threshold = 11
elif trial_id in [2, 3]:
    n_class_threshold = 25
r_agg = 7 # initial aggregation radius for roi in pixel locations
buffer = 45 # avoid that area determination selects a window that is too close to the border

# load scene instance for this slice:
DATADIR = fr"C:\Users\aroessel\Desktop\real_data\05_analyses\slice-{str(SLICE).zfill(3)}\trial-{str(trial_id).zfill(3)}"
with open(DATADIR + "\scene.pkl", 'rb') as f:
    scene = pickle.load(f)

# load model
model_area = keras.models.load_model(MODEL_DIR_PARAM)

for CYCLE in CYCLES:
    print("\n", fr"Started with Cycle: {CYCLE}")
    # load data
    image = scene.images[CYCLE].copy()
```

```python
    prediction = pickle.load(open(DATADIR + fr"\cycle_predictions\cycle-{str(CYCLE).zfill(3)}.pkl", 'rb'))
    pixel_locations = np.array(prediction.pixel_locations.copy(), dtype=np.float64)
    pixel_aggregation_map = prediction.pixel_locations.copy()
    abs_locations = prediction.abs_locations.copy()

    # prepare output data structures
    pos_aggregated = pixel_locations.copy() # in this array, pixel_locations will be aggregated in-place
    cycle_out = Cycle(scene.shape, scene.images[CYCLE].copy())

    # identify all local maxima
    local_maxima = []
    for i in range(1, pixel_locations.shape[0] - 1):
        for j in range(1, pixel_locations.shape[1] - 1):
            local_region = pixel_locations[i-1:i+2, j-1:j+2]
            max_indices = np.unravel_index(np.argmax(local_region, axis=None), local_region.shape)
            if max_indices == (1, 1):
                local_maxima.append((i, j, pixel_locations[i, j]))
    local_maxima = np.array(local_maxima, dtype=np.int16)

    # sort list of local maxima from large to small
    if local_maxima.size > 0:
        local_maxima = local_maxima[np.argsort(local_maxima[:, 2])]
    local_maxima = local_maxima[::-1]
    local_maxima = list(local_maxima)

    # AGGREGATE BLISTERS and calculate position with standard deviations
    while len(local_maxima) > 0:
        # get largest maximum
        maxi = local_maxima[0]
        i = maxi[0]
        j = maxi[1]

        # check whether local maximum is too close to borders (a 64*64 frame around the position is needed for parameter determination). Therefore, a border range of 32 + extra tolerance due to localization inside the r_agg is excluded from analysis.
        if i not in range(buffer, scene.shape[0] - buffer) or j not in range(buffer, scene.shape[1] - buffer):
            local_maxima.pop(0)
            continue

        # check whether this maximum was consumed by a higher-order maximum
        if pixel_aggregation_map[i, j] == 0:
            local_maxima.pop(0)
            continue

        # get localizations inside roi for first area analysis
        roi_list = []
        for k in range(i - r_agg, i + r_agg + 1):
            for l in range(j - r_agg, j + r_agg + 1):
                for n in range(pixel_aggregation_map[k, l]):
                    loc = np.array([l, k], dtype=np.float64)
                    dist = round(np.sqrt((loc[1] - i) ** 2 + (loc[0] - j) ** 2), 0)
                    if dist <= r_agg:
                        roi_list.append(loc)

        # calculate mean position, standard deviation and number of localizations in roi
        abs_x, abs_y, n_sum, sd_x, sd_y = analyse_roi(roi_list)

        # get image section
        y_start = int(round(abs_y - WINDOW_SHAPE[0] / 2, 0))
        y_end = y_start + WINDOW_SHAPE[0]
        x_start = int(round(abs_x - WINDOW_SHAPE[1] / 2, 0))
        x_end = x_start + WINDOW_SHAPE[1]

        image_section = image[y_start:y_end, x_start:x_end].copy()
        image_section_tf = np.expand_dims(image_section, (0, 3))  # this converts the image to tensorflow format

        # predict area
        area = model_area.predict(image_section_tf)
        area = max([area, 0]) # necessary because in some seldomcases, the area turns out to be negative
        r_agg_new = max([int(np.sqrt(area / np.pi)), 5])

        # get localizations inside roi for further analysis and deactivate them
        roi_list = []
        # use new x, y that were updated in first iteration:
        y_new = int(round(abs_y, 0))
        x_new = int(round(abs_x, 0))
        r_agg_new_x = int(round(r_agg_new * 1.5, 0))# in x-direction r_agg is scaled because blisters are typically wider than high
        for k in range(y_new - r_agg_new, y_new + r_agg_new + 1):
            for l in range(x_new - r_agg_new_x, x_new + r_agg_new_x + 1):
                for n in range(pixel_aggregation_map[k, l]):
                    loc = np.array([l, k], dtype=np.float64)
                    dist = round(np.sqrt((loc[1] - y_new) ** 2 + (loc[0] - x_new) ** 2), 0)
                    if dist <= r_agg_new:
                        roi_list.append(loc)

                # delete identifications from map
                pixel_aggregation_map[k, l] = 0

        # calculate mean position, standard deviation and number of localizations in roi
        abs_x, abs_y, n_sum, sd_x, sd_y = analyse_roi(roi_list)
        print(n_sum)

        # identify blister and check threshold for n_sum and calculate area of blister candidate
        if n_sum >= n_class_threshold:
            # get image section
            y_start = int(round(abs_y - WINDOW_SHAPE[0] / 2, 0))
            y_end = y_start + WINDOW_SHAPE[0]
            x_start = int(round(abs_x - WINDOW_SHAPE[1] / 2, 0))
            x_end = x_start + WINDOW_SHAPE[1]

            image_section = image[y_start:y_end, x_start:x_end].copy()
            image_section_tf = np.expand_dims(image_section, (0, 3))  # this converts the image to tensorflow format

            # predict area
            area = model_area.predict(image_section_tf)

            # add blister to scene
            cycle_out.add_Blister((abs_x, abs_y), n_sum, area, (sd_x, sd_y))
            print("Added a blister!!! \n")

        # delete current local maximum from list
        local_maxima.pop(0)

    # write output to disk

    with open(DATADIR + fr"\cycle_identifications\cycle-{str(CYCLE).zfill(3)}.pkl", 'wb') as f:
        pickle.dump(cycle_out, f)

    print(cycle_out.blisters.keys())
```

Listing 9: *This script interprets the predictions of the CNNs made with the script given in the preceding code listing, defines the actual blister identifications and creates a 'Blister' instance accordingly.*

```
import pickle
from configs_and_resources import *

# choose cycles to be analyzed
CYCLES = []
for i in range(18, 100):
    CYCLES.append(i)

# load scene instance for this slice:
DATADIR = fr"C:\Users\aroessel\Desktop\real_data\05_analyses\slice-{str(SLICE).zfill(3)}\trial-{str(trial_id).zfill(3)}"
with open(DATADIR + "\scene.pkl", 'rb') as f:
    scene = pickle.load(f)

# reset tracking analysis
scene.reset()

for CYCLE in CYCLES:
    print("\n", fr"Started with Cycle: {CYCLE}")

    # grab current cycle's identifications
    with open(DATADIR + fr"\cycle_identifications\cycle-{str(CYCLE).zfill(3)}.pkl", 'rb') as f:
        cycle = pickle.load(f)

    # loop over new blisters to be added
    for key in cycle.blisters.keys():
        new_blister = cycle.blisters[f'{key}']

        # load new blister parameters
        position = new_blister.positions[0]
        n_classifications = new_blister.n_classifications[0]
        area = new_blister.areas[0]
        sds = new_blister.sds[0]
```

```python

        was_old = False  # switch between update old blister and create new blister
        radius = np.sqrt(area * 0.5 / np.pi)

        # UPDATE OLD: loop over old blisters and update when appropriate
        for key_scene in scene.blisters.keys():
            old_blister = scene.blisters[key_scene]
            if old_blister.n_classifications[CYCLE - 1] == 0:
                continue
            x_diff = old_blister.positions[CYCLE - 1, 0] - position[0]
            x_diff = x_diff * 2/3       # reduce contribution in x direction due to angle of
    sight
            y_diff = old_blister.positions[CYCLE - 1, 1] - position[1]
            dist = np.sqrt(x_diff ** 2 + y_diff ** 2)

            if dist < radius:
                old_blister.add_cycle(CYCLE, position, n_classifications, area, sds)
                output = fr"Updated blister with key {key_scene}"
                was_old = True
                break

        # CREATE NEW: if no old blister was updated, create a new one
        if not was_old:
            output = scene.add_Blister(CYCLE, position, n_classifications, area, sds)

# write output to disk
with open(DATADIR + fr"\scene_tracked.pkl", 'wb') as f:
    pickle.dump(scene, f)
    print("Saved final results.")
```

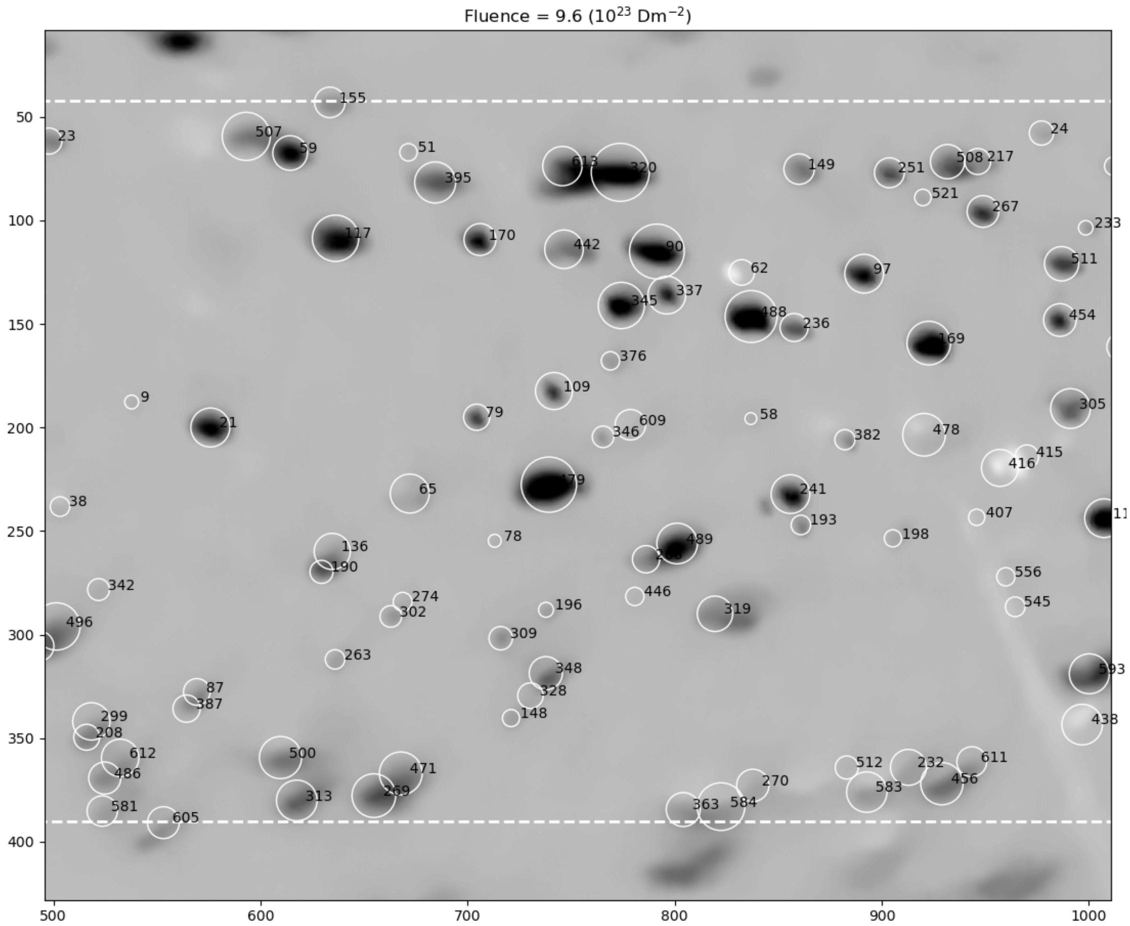
Listing 10: *This script implements the tracking of individual blisters across mutliple image cylces.*

```python
import pickle
from configs_and_resources import *

# load scene instance for this slice:
DATADIR = fr"C:\Users\aroessel\Desktop\real_data\05_analyses\slice-{str(SLICE).zfill(3)}\trial-{
    str(trial_id).zfill(3)}"
with open(DATADIR + "\scene_tracked.pkl", 'rb') as f:
    scene = pickle.load(f)

print(len(scene.blisters.keys()))

keys = list(scene.blisters.keys())
lifetimes = []
deleted = {}

for key in keys:
    blister = scene.blisters[fr"{key}"]
    lifetime = blister.get_lifetime()
    # lifetimes.append(lifetime)
    if lifetime == 0:
        deleted[f'{key}'] = blister
        del scene.blisters[fr"{key}"]
        # print(fr"poped key {key}")

print(len(scene.blisters.keys()))

scene.deleted_blisters = deleted

# write output to disk
with open(DATADIR + fr"\scene_final.pkl", 'wb') as f:
    pickle.dump(scene, f)
    print("Saved final results.")
```

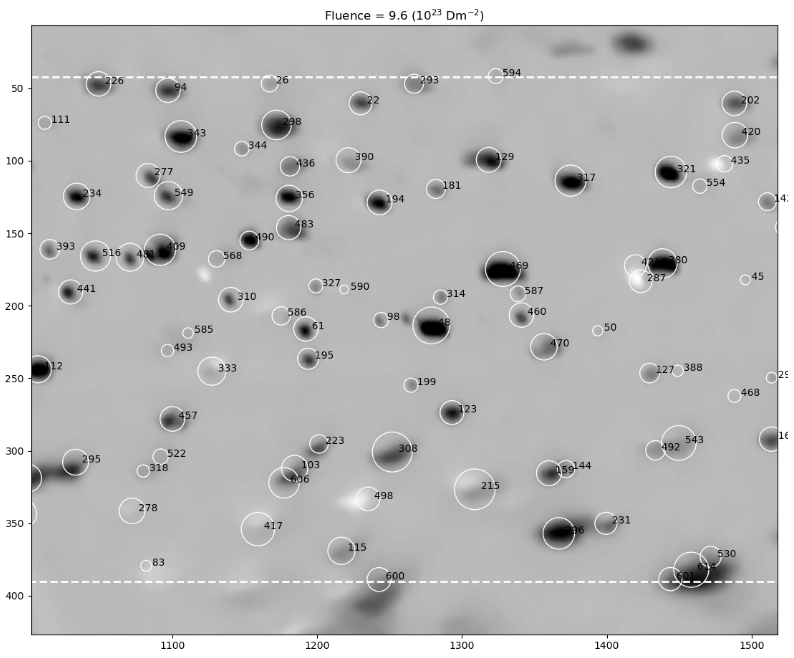
Listing 11: *This script implements all applying postprocessing steps.*

F Image Sections Considered for Assessment of Accuracy

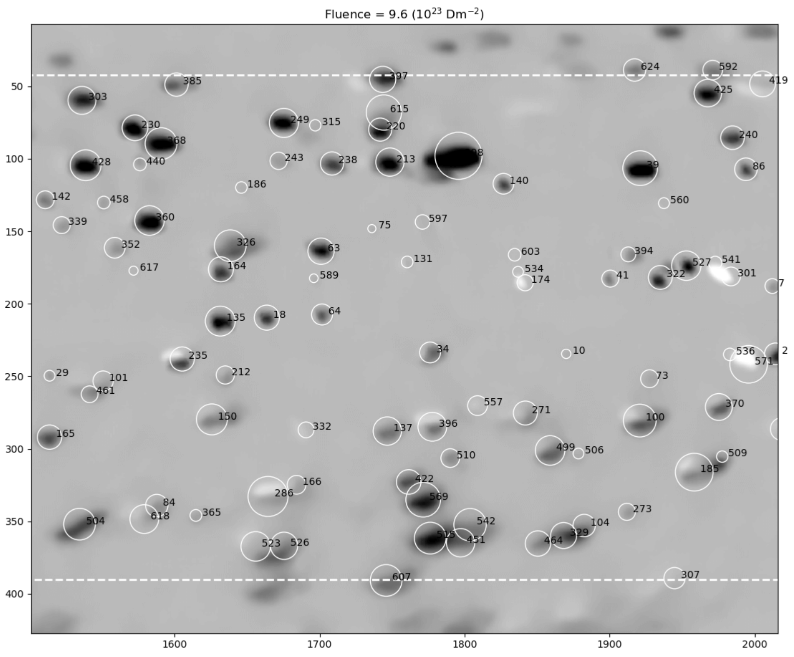
These image sections were used for the analysis of the accuracy of the identifications. The respective image cycle corresponds to a fluence of 9.6×10^{23} Dm^{-2}. A total of 221 blister instances were considered. The white dashed line indicates the area in which blisters can get identified.



I


Fluence = 9.6 (10^{23} Dm^{-2})
50
100
150
200
250
300
350
400
1100
1200
1300
1400
1500

II


Fluence = 9.6 (10^{23} Dm^{-2})
50
100
150
200
250
300
350
400
1600
1700
1800
1900
2000

III

Bibliography

[1] John Cook et al. »Consensus on consensus: a synthesis of consensus estimates on human-caused global warming«. In: *Environmental Research Letters* 11 (2016).

[2] Mengpin Ge Johannes Friedrich and Andrew Pickens. *World's Top 10 Emitters, and How They've Changed.* online. Apr. 2017. URL: https://www.wri.org/blog/2017/04/interactive-chart-explains-worlds-top-10-emitters-and-how-theyve-changed.

[3] Christoph Frei et al. *World energy scenarios: Composing energy futures to 2050.* Tech. rep. Conseil Francais de l'energie, 2013.

[4] Joachim Roth et al. »Recent analysis of key plasma wall interactions issues for ITER«. In: *Journal of Nuclear Materials* 390 (2009), pp. 1–9.

[5] J.B. Condon and T. Schober. »Hydrogen bubbles in metals«. In: *Journal of Nuclear Materials 207.* 1993. URL: https://www.sciencedirect.com/science/article/pii/002231159390244S?via%3Dihub.

[6] A Manhard, T Schwarz-Selinger and W Jacob. »Quantification of the deuterium ion fluxes from a plasma source«. In: *Plasma Sources Science and Technology* 20.1 (Jan. 2011). DOI: 10.1088/0963-0252/20/1/015010.

[7] Alexander van Roessel. *Automatic Blister Detection.* 2020. URL: https://gitlab.mpcdf.mpg.de/aroessel/automatic-blister-detection.

[8] JE Lennard-Jones. »Processes of adsorption and diffusion on solid surfaces«. In: *Transactions of the Faraday Society* 28 (1932), pp. 333–359.

[9] Y Ebisuzaki and M O'keeffe. »The solubility of hydrogen in transition metals and alloys«. In: *Progress in Solid State Chemistry* 4 (1967), pp. 187–211.

[10] D Zamir. »Nuclear-magnetic-resonance study of hydrogen alloying in the early transition metals (Group VB)«. In: *Physical Review* 140.1A (1965), A271.

[11] Mikhail Zibrov. »The influence of radiation, mechanical, and plasma-induced damage on deuterium retention in tungsten«. PhD thesis. Gent University, 2018.

[12] A Sieverts and E Jurisch. »Platin, Rhodium und Wasserstoff«. In: *Berichte der deutschen chemischen Gesellschaft* 45.1 (1912), pp. 221–229.

[13] Yuh Fukai. *The metal-hydrogen system: basic bulk properties.* Vol. 21. Springer Science & Business Media, 2006.

[14] W Moeller and J Roth. »Implantation, retention and release of hydrogen isotopes in solids«. In: *Physics of plasma-wall interactions in controlled fusion.* Springer, 1986, pp. 439–494.

[15] Yu-Wei You et al. »Dissolving, trapping and detrapping mechanisms of hydrogen in bcc and fcc transition metals«. In: *AIP advances* 3.1 (2013).

[16] H Wipf. »Solubility and diffusion of hydrogen in pure metals and alloys«. In: *Physica Scripta* 2001.T94 (2001), p. 43.

[17] Martin Balden et al. »D2 gas-filled blisters on deuterium-bombarded tungsten«. In: *Journal of Nuclear Materials* 414.1 (July 2011), pp. 69–72. DOI: `10.1016/j.jnucmat.2011.04.031`.

[18] S Lindig et al. »Subsurface morphology changes due to deuterium bombardment of tungsten«. In: *Physica Scripta* 2009.T138 (2009).

[19] S Lindig et al. »Sub-surface structures of ITER-grade W (Japan) and recrystallized W after ITER-similar low-energy and high-flux D plasma loadings«. In: *Physica Scripta* 2011.T145 (2011), p. 014039.

[20] W Mo Shu, G-N Luo and T Yamanishi. »Mechanisms of retention and blistering in near-surface region of tungsten exposed to high flux deuterium plasmas of tens of eV«. In: *Journal of Nuclear Materials* 367 (2007), pp. 1463–1467.

[21] OV Ogorodnikova, J Roth and M Mayer. »Ion-driven deuterium retention in tungsten«. In: *Journal of Applied Physics* 103.3 (2008).

[22] Armin Manhard. »Deuterium inventory in tungsten after plasma exposure: a microstructural survery«. PhD thesis. Augsburg University, 2012. URL: `https://pure.mpg.de/rest/items/item_2146108/component/file_2146107/content`.

[23] M Balden et al. »Surface morphology and deuterium retention of tungsten after low-and high-flux deuterium plasma exposure«. In: *Nuclear Fusion* 54.8 (2014).

[24] Armin Manhard, Martin Balden and Udo von Toussaint. »Blister formation on rough and technical tungsten surfaces exposed to deuterium plasma«. In: *Nuclear Fusion* 57.12 (Sept. 2017). DOI: `10.1088/1741-4326/aa82c8`.

[25] A Manhard et al. »Influence of the microstructure on the deuterium retention in tungsten«. In: *Journal of nuclear materials* 415.1 (2011), S632–S635.

[26] Frank M Richter et al. »Isotope fractionation by chemical diffusion between molten basalt and rhyolite«. In: *Geochimica et Cosmochimica Acta* 67.20 (2003), pp. 3905–3923.

[27] A Manhard et al. »Statistical analysis of blister bursts during temperature-programmed desorption of deuterium-implanted polycrystalline tungsten«. In: *Physica Scripta* 2011.T145 (2011).

[28] RA Causey et al. »Defects in tungsten responsible for molecular hydrogen isotope retention after exposure to low energy plasmas«. In: *Journal of nuclear materials* 390 (2009), pp. 717–720.

[29] V Kh Alimov et al. »Temperature dependence of surface topography and deuterium retention in tungsten exposed to low-energy, high-flux D plasma«. In: *Journal of Nuclear Materials* 417.1-3 (2011), pp. 572–575.

[30] A Manhard et al. »Microstructure and defect analysis in the vicinity of blisters in polycrystalline tungsten«. In: *Nuclear Materials and Energy* 12 (2017), pp. 714–719.

[31] Peter Gumbsch et al. »Controlling factors for the brittle-to-ductile transition in tungsten single crystals«. In: *Science* 282.5392 (1998), pp. 1293–1295.

[32] J Bauer et al. »Influence of near-surface blisters on deuterium transport in tungsten«. In: *Nuclear Fusion* 57.8 (2017), p. 086015. DOI: `https://iopscience.iop.org/article/10.1088/1741-4326/aa7212`. URL: `https://iopscience.iop.org/article/10.1088/1741-4326/aa7212`.

[33] Grégoire Montavon, Wojciech Samek and Klaus-Robert Müller. »Methods for interpreting and understanding deep neural networks«. In: *Digital Signal Processing* 73 (2018), pp. 1–15.

[34] Nikolaos Pappas and Andrei Popescu-Belis. »Human versus machine attention in document classification: A dataset with crowdsourced annotations«. In: *Proceedings of The Fourth International Workshop on Natural Language Processing for Social Media.* 2016, pp. 94–100.

[35] Grace Lindsay. »Convolutional neural networks as a model of the visual system: past, present, and future«. In: *Journal of Cognitive Neuroscience* (2020), pp. 1–15.

[36] Dorian Pyle and Cristina San José. »An executive's guide to machine learning«. In: *McKinsey Quarterly* (2015). URL: `https://www.mckinsey.com/industries/technology-media-and-telecommunications/our-insights/an-executives-guide-to-machine-learning`.

[37] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.* O'Reilly Media, 2019.

[38] Dana Harry Ballard and Christopher M Brown. *Computer vision.* Prentice Hall, 1982. URL: http://homepages.inf.ed.ac.uk/rbf/BOOKS/BANDB/Ballard__D._and_Brown__C._M.__1982__Computer_Vision.pdf.

[39] Athanasios Voulodimos et al. »Deep learning for computer vision: A brief review«. In: *Computational intelligence and neuroscience* 2018 (2018).

[40] Fei-Fei Li, Justin Johnson and Serena Yeung. *Convolutional Neural Networks for Visual Recognition.* online. 2017. URL: http://cs231n.stanford.edu/2017/.

[41] Li Liu et al. »Deep learning for generic object detection: A survey«. In: *International journal of computer vision* 128.2 (2020), pp. 261–318.

[42] Farhana Sultana, Abu Sufian and Paramartha Dutta. »A review of object detection models based on convolutional neural network«. In: *Intelligent Computing: Image Processing Based Applications.* Springer, 2020, pp. 1–16.

[43] Geoffrey E. Hinton, Simon Osindero and Yee-Whye Teh. »A Fast Learning Algorithm for Deep Belief Nets«. In: *Neural Computation* 18.7 (July 2006), pp. 1527–1554. DOI: 10.1162/neco.2006.18.7.1527. URL: https://www.mitpressjournals.org/doi/abs/10.1162/neco.2006.18.7.1527.

[44] B.J. Copeland. *Artificial intelligence.* online. Mar. 2020. URL: https://www.britannica.com/technology/artificial-intelligence.

[45] Warren S. McCulloch and Walter Pitts. »A logical calculus of the ideas immanent in nervous activity«. In: *The Bulletin of Mathematical Biophysics* 5.4 (Dec. 1943), pp. 115–133. DOI: 10.1007/bf02478259.

[46] Frank Rosenblatt. *Principles of neurodynamics. perceptrons and the theory of brain mechanisms.* Tech. rep. Cornell Aeronautical Lab Inc Buffalo NY, 1961. URL: https://apps.dtic.mil/docs/citations/AD0256582.

[47] David E Rumelhart, Geoffrey E Hinton and Ronald J Williams. *Learning internal representations by error propagation.* Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science, 1985. URL: https://apps.dtic.mil/docs/citations/ADA164453.

[48] Bernard Widrow and Marcian E Hoff. *Adaptive switching circuits.* Tech. rep. Stanford Univ Ca Stanford Electronics Labs, 1960. URL: https://apps.dtic.mil/dtic/tr/fulltext/u2/241531.pdf.

[49] Christopher M Bishop et al. *Neural networks for pattern recognition.* Oxford university press, 1995. URL: https://pdfs.semanticscholar.org/f105/04f73ce5916c6283f782129467d6156fd9c6.pdf.

[50] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para.* Cornell Aeronautical Laboratory, 1957.

[51] Kevin Jarrett et al. »What is the best multi-stage architecture for object recognition?« In: *2009 IEEE 12th international conference on computer vision*. IEEE. 2009, pp. 2146–2153.

[52] Vinod Nair and Geoffrey E Hinton. »Rectified linear units improve restricted boltzmann machines«. In: *ICML*. 2010.

[53] Xavier Glorot, Antoine Bordes and Yoshua Bengio. »Deep sparse rectifier neural networks«. In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 2011, pp. 315–323.

[54] George E Dahl, Tara N Sainath and Geoffrey E Hinton. »Improving deep neural networks for LVCSR using rectified linear units and dropout«. In: *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE. 2013, pp. 8609–8613.

[55] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[56] Kurt Hornik. »Approximation capabilities of multilayer feedforward networks«. In: *Neural networks* 4.2 (1991), pp. 251–257. DOI: `https://doi.org/10.1016/0893-6080(91)90009-T`.

[57] Ken-Ichi Funahashi. »On the approximate realization of continuous mappings by neural networks«. In: *Neural networks* 2.3 (1989), pp. 183–192.

[58] George Cybenko. »Approximation by superpositions of a sigmoidal function«. In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.

[59] Kurt Hornik, Maxwell Stinchcombe, Halbert White et al. »Multilayer feedforward networks are universal approximators.« In: *Neural networks* 2.5 (1989), pp. 359–366.

[60] Allan Pinkus. »Approximation theory of the MLP model in neural networks«. In: *Acta numerica* 8.1 (1999), pp. 143–195.

[61] Richard Lippmann. »An introduction to computing with neural nets«. In: *IEEE Assp magazine* 4.2 (1987), pp. 4–22.

[62] Moshe Leshno et al. »Multilayer feedforward networks with a nonpolynomial activation function can approximate any function«. In: *Neural networks* 6.6 (1993), pp. 861–867. DOI: `https://doi.org/10.1016/S0893-6080(05)80131-5`.

[63] Xavier Glorot and Yoshua Bengio. »Understanding the difficulty of training deep feedforward neural networks«. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010, pp. 249–256.

[64] IJ Good. »Some terminology and notation in information theory«. In: *Proceedings of the IEE-Part C: Monographs* 103.3 (1956), pp. 200–204.

[65] Patrice Y Simard, David Steinkraus, John C Platt et al. »Best practices for convolutional neural networks applied to visual document analysis.« In: *Icdar*. Vol. 3. 2003. 2003.

[66] David E Rumelhart, Geoffrey E Hinton and Ronald J Williams. »Learning representations by back-propagating errors«. In: *nature* 323.6088 (1986), pp. 533–536.

[67] Augustin Cauchy. »Méthode générale pour la résolution des systemes d'équations simultanées«. In: *Comp. Rend. Sci. Paris* 25.1847 (1847), pp. 536–538. URL: `https://cs.uwaterloo.ca/~y328yu/classics/cauchy-en.pdf`.

[68] G Temple. »The general theory of relaxation methods applied to linear systems«. In: *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences* 169.939 (1939), pp. 476–500.

[69] Haskell B Curry. »The method of steepest descent for non-linear minimization problems«. In: *Quarterly of Applied Mathematics* 2.3 (1944), pp. 258–261.

[70] Diederik P Kingma and Jimmy Ba. »Adam: A method for stochastic optimization«. In: *arXiv preprint arXiv:1412.6980* (2014).

[71] Yann LeCun et al. »Backpropagation applied to handwritten zip code recognition«. In: *Neural computation* 1.4 (1989), pp. 541–551.

[72] David C Plaut et al. »Experiments on Learning by Back Propagation.« In: (1986).

[73] Y Nesterov. »A method for solving the convex programming problem with convergence rate $O(1/k^2)$«. In: *Soviet Math. Dokl.* Vol. 27. 2. 1983, pp. 372–376.

[74] Ning Qian. »On the momentum term in gradient descent learning algorithms«. In: *Neural networks* 12.1 (1999), pp. 145–151.

[75] John Duchi, Elad Hazan and Yoram Singer. »Adaptive subgradient methods for online learning and stochastic optimization.« In: *Journal of machine learning research* 12.7 (2011).

[76] Matthew D Zeiler. »Adadelta: an adaptive learning rate method«. In: *arXiv preprint arXiv:1212.5701* (2012).

[77] Sashank J Reddi, Satyen Kale and Sanjiv Kumar. »On the convergence of adam and beyond«. In: *arXiv preprint arXiv:1904.09237* (2019).

[78] Sebastian Ruder. »An overview of gradient descent optimization algorithms«. In: *arXiv preprint arXiv:1609.04747* (2016).

[79] Tomaso Poggio, Vincent Torre and Christof Koch. »Computational vision and regularization theory«. In: *nature* 317.6035 (1985), pp. 314–319.

[80] Federico Girosi, Michael Jones and Tomaso Poggio. »Regularization theory and neural networks architectures«. In: *Neural computation* 7.2 (1995), pp. 219–269.

[81] Udo v Toussaint, Silvio Gori and Volker Dose. »Invariance priors for Bayesian feed-forward neural networks«. In: *Neural Networks* 19.10 (2006), pp. 1550–1557.

[82] Geoffrey E Hinton et al. »Improving neural networks by preventing co-adaptation of feature detectors«. In: *arXiv preprint arXiv:1207.0580* (2012).

[83] Nitish Srivastava et al. »Dropout: a simple way to prevent neural networks from overfitting«. In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[84] Yann LeCun, Yoshua Bengio et al. »Convolutional networks for images, speech, and time series«. In: *The handbook of brain theory and neural networks* 3361.10 (1995), p. 1995.

[85] Kunihiko Fukushima and Sei Miyake. »Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition«. In: *Competition and cooperation in neural nets*. Springer, 1982, pp. 267–285.

[86] Yann LeCun et al. »Handwritten digit recognition with a back-propagation network«. In: *Advances in neural information processing systems*. 1990, pp. 396–404.

[87] Yann LeCun et al. »Gradient-based learning applied to document recognition«. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[88] David H Hubel and Torsten N Wiesel. »Receptive fields of single neurones in the cat's striate cortex«. In: *The Journal of physiology* 148.3 (1959), p. 574.

[89] Yann LeCun, Yoshua Bengio and Geoffrey Hinton. »Deep learning«. In: *nature* 521.7553 (2015), pp. 436–444.

[90] Alex Krizhevsky, Ilya Sutskever and Geoffrey E Hinton. »Imagenet classification with deep convolutional neural networks«. In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[91] Jason Yosinski et al. »How transferable are features in deep neural networks?« In: *Advances in neural information processing systems*. 2014, pp. 3320–3328.

[92] Dominik Scherer, Andreas Müller and Sven Behnke. »Evaluation of pooling operations in convolutional architectures for object recognition«. In: *International conference on artificial neural networks*. Springer. 2010, pp. 92–101.

[93] Isma Hadji and Richard P Wildes. »What do we understand about convolutional networks?« In: *arXiv preprint arXiv:1803.08834* (2018).

[94] Sumit Saha. *CNN setup for the classification of handwritten digits.* online. Dec. 2018. URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[95] Julian D Olden and Donald A Jackson. »Illuminating the “black box”: a randomization approach for understanding variable contributions in artificial neural networks«. In: *Ecological modelling* 154.1-2 (2002), pp. 135–150.

[96] Jason Yosinski et al. »Understanding neural networks through deep visualization«. In: *arXiv preprint arXiv:1506.06579* (2015).

[97] Matthew D Zeiler and Rob Fergus. »Visualizing and understanding convolutional networks«. In: *European conference on computer vision.* Springer. 2014, pp. 818–833.

[98] Jost Tobias Springenberg et al. »Striving for simplicity: The all convolutional net«. In: *arXiv preprint arXiv:1412.6806* (2014).

[99] Pablo d’Angelo. *Hugin - Panorama photo stitcher.* online. Sept. 2020. URL: http://hugin.sourceforge.net/.

[100] Pablo d’Angelo. *Hugin Tutorials.* online. Sept. 2020. URL: http://hugin.sourceforge.net/tutorials/index.shtml.

[101] Pablo d’Angelo. *hugin source code documentation.* online. Sept. 2020. URL: http://hugin.sourceforge.net/docs/html/.

[102] Greg Lehey. *Greg’s aligning images with Hugin.* online. Sept. 2020. URL: http://hugin.sourceforge.net/docs/html/.

[103] Antoni Buades, Bartomeu Coll and Jean-Michel Morel. »Non-local means denoising«. In: *Image Processing On Line* 1 (2011), pp. 208–212.

[104] Olli-Pekka Heinisuo. *OpenCV Python download page.* online. Oct. 2020. URL: https://pypi.org/project/opencv-python/.

[105] David Kirk Buck and Aaron A. Collins. *POV-Ray download page.* online. Persistence of Vision Raytracer Pty. Ltd, Oct. 2020. URL: https://www.povray.org/download/.

[106] David Kirk Buck and Aaron A. Collins. *POV-Ray Online Documentation.* online. Persistence of Vision Raytracer Pty. Ltd, Oct. 2020. URL: https://www.povray.org/documentation/3.7.0/.

[107] Martın Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.* Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[108] Google Brain team. *Tensorflow Case Studies.* online. Oct. 2020. URL: https://www.tensorflow.org/about/case-studies.

[109] Francois Chollet et al. *Keras*. 2015. URL: `https://github.com/fchollet/keras`.

[110] Marc Claesen and Bart De Moor. »Hyperparameter search in machine learning«. In: *arXiv preprint arXiv:1502.02127* (2015).

[111] Jonas Mockus. *Bayesian approach to global optimization: theory and applications*. Vol. 37. Springer Science & Business Media, 2012.

[112] Jasper Snoek, Hugo Larochelle and Ryan P Adams. »Practical bayesian optimization of machine learning algorithms«. In: *Advances in neural information processing systems*. 2012, pp. 2951–2959.

[113] Francois Chollet et al. *Keras*. 2015. URL: `https://keras-team.github.io/keras-tuner/documentation/tuners/#bayesianoptimization-class`.

[114] Karen Simonyan and Andrew Zisserman. »Very deep convolutional networks for large-scale image recognition«. In: *arXiv preprint arXiv:1409.1556* (2014).

[115] Katarzyna Janocha and Wojciech Marian Czarnecki. »On loss functions for deep neural networks in classification«. In: *Schedae Informaticae* 25 (2017). DOI: `10.4467/20838476SI.16.004.6185`.

[116] L Gao et al. »Suppression of hydrogen-induced blistering of tungsten by pre-irradiation at low temperature«. In: *Nuclear Fusion* 54.12 (2014), p. 122003.

[117] RW Thomas. »Nearest Neighbor Distances in an Infinite Entropic Point Pattern«. In: *Geographical Analysis* 9.4 (1977), pp. 409–418.

Acknowledgements

Firstly, I would like to express my sincere gratitude to both of my supervisors, **Dr. Armin Manhard** and **Dr. Udo von Toussaint**. Both of them have provided great support and have been instructive examples for me in terms of scientific work. They were always eager to support me and expand my knowledge.

Also, I would like to thank **Prof. Dr. Ulrich Stroth** for agreeing to supervise my master thesis.

I am also very grateful to the other students, in particular **Florian Günzkofer**, who provided great feedback and company throughout my entire stay at the Max-Planck-Institut für Plasmaphysik (IPP).

Finally, I thank all the other colleagues at IPP and the institute itself for offering a great environment for fusion research.